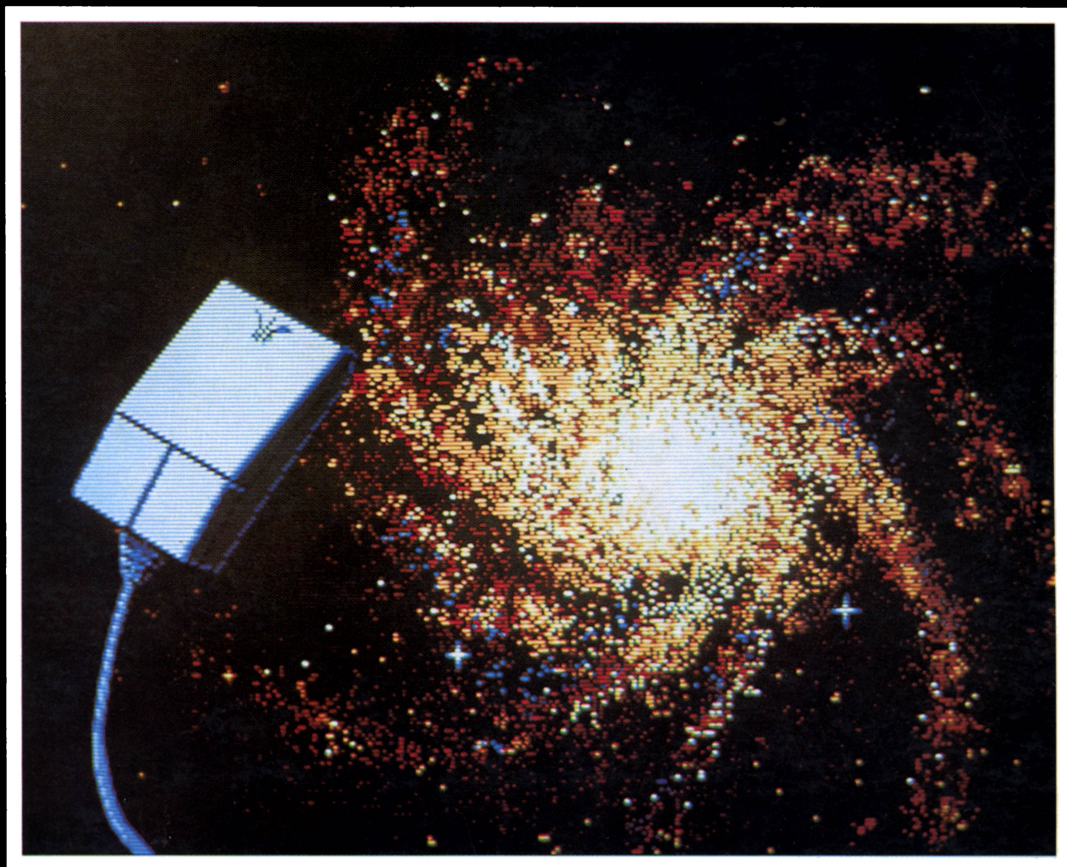
ATARI® ST™

programmazione in BASIC



tecniche nuove

TIM KNIGHT

Tim Knight

ATARI® ST™
programmazione in BASIC

Tim Knight

ATARI® ST™
programmazione in BASIC

tecniche nuove

Questo libro è dedicato a mio padre, Delos Knight Jr., che mi ha comprato il primo computer e che da allora mi ha aiutato e incoraggiato.

EDIZIONE ITALIANA
Tim Knight - Understanding Atari® ST™ BASIC Programming

Illustrazione della copertina di Thomas Ingalls + Associates.

La fotografia dell'Atari 1040ST della Figura 1.1 compare per concessione dell'Atari Corp.
Tutte le altre fotografie sono di Paul Herzoff.

ISBN 88 7081 325 8

Fotocomposizione: Grafica Quadrifoglio, Milano
Stampa: Polver, Milano
(0970)

INDICE

Ringraziamenti

Vorrei ringraziare ognuna delle persone e delle aziende seguenti per l'aiuto prestatomi mentre scrivevo il libro; hanno reso il mio lavoro molto più facile.

All'Atari mi hanno fornito informazioni e materiali essenziali:

Kim Barney
Michael Katz
Connie McBride
Arnold Waldstein

Alla Sybex hanno dato il loro notevole contributo al libro:

Chuck Ackermann, redazione
Lisa Amon, grafica
Suzy Anger, correzione bozze
Dave Clark, elaborazione testi
Jim Compton, redazione tecnica
Joel Kroman, supervisione tecnica
Rudolph Langer, coordinamento editoriale
Karl Ray, direzione editoriale
Olivia Shinomoto, elaborazione testi
Dan Tauber, supervisione tecnica
Brenda Walker, composizione

Hanno inoltre contribuito in vari modi:

Naomi Brooksbank della B & C Computervision of Sunnyvase, California
Tanya Kucak, redazione, e William Naylor
della EXC Computer Co. of Walnut Creek, California

INTRODUZIONE

I computer Atari 520ST e 1040ST mantengono le promesse fatte dai loro costruttori: “Potenza e prezzo”, infatti per circa un milione potete acquistare un eccellente computer con una grafica a colori superiore, capacità di emissione di suoni e musica a tre voci, di elaborazione veloce e di memorizzazione massiccia ad alta qualità. Il modo più efficace per sfruttare tutte queste potenzialità è il linguaggio di programmazione BASIC ST, contenuto nel dischetto ST Language che accompagna il computer.

Se possedete o potete disporre di un computer ST, questo libro vi aiuterà ad impararne la programmazione e vi permetterà di esplorare, e infine di controllare, le sue caratteristiche avanzate. Per usare questo libro non è necessario che conosciate già il BASIC (o nessun altro linguaggio di programmazione), perché cominceremo dall'inizio. Se invece avete qualche familiarità con il BASIC, saprete che le varie versioni del linguaggio differiscono in qualche misura; questo libro tratta specificamente la versione Atari ST. Sia che usiate il 520ST o il 1040ST, potete diventare buoni programmatori in BASIC ST in tempi piuttosto brevi, e non escludo che vi possiate anche divertire parecchio nel processo di apprendimento.

HARDWARE E SOFTWARE

Le informazioni contenute in questo libro sono applicabili a tutti i modelli di computer ST. Le differenze tra i vari modelli sono spiegate nel Capitolo 1; in sintesi il 1040ST dispone di più memoria del 520ST ed ha il drive incorporato, anziché esterno. In aggiunta, i computer 520 ST venduti a partire dal febbraio 1986 sono diversi dalle versioni precedenti dello stesso modello. I 520ST più recenti contengono un sistema operativo in memoria permanente, invece per le versioni precedenti esso è fornito separatamente, su disco.

Per procedere con gli esempi di programmazione di questo libro vi serve un computer ST ed il disco ST Language che lo accompagna. Se avete un vecchio 520ST, vi dovrete servire del disco ST System, che accompagnava il computer. Questo è tutto, non serve altro.

COSA CONTIENE IL LIBRO?

Diamo una panoramica dei capitoli che compongono questo testo.

Il Capitolo 1, *Facciamo la conoscenza del ST*, vi presenta i diversi componenti del ST e le modalità per l'uso della scrivania GEM.

Il Capitolo 2, *Elementi fondamentali del BASIC*, vi mostra i comandi principali necessari a costruire un programma. Lo troverete utile se il BASIC ST vi è totalmente nuovo o se, pur capendo il BASIC, volete acquisire familiarità con le caratteristiche specifiche del BASIC ST, oltre che con i numeri di linea, i loop e la logica. Imparerete anche i potenti comandi di editing del BASIC ST.

Il Capitolo 3, *Parole e numeri*, è dedicato alle funzioni di testo e matematiche. Poiché l'Atari dispone di un gran numero di parole chiave in quest'area, ne farò un elenco e descriverò ciascuna di esse dettagliatamente.

Il Capitolo 4 è intitolato *Grafica ST.* La grafica sull'Atari ST è facile da programmare e spettacolare da guardare e una volta che saprete come programmarla con il BASIC ST, avrete la tentazione di passare giorni interi a sperimentarla. Dopo questo capitolo, saprete creare con facilità cerchi, linee, poligoni e oggetti tridimensionali. Imparerete anche come ottenere ciascuno dei 520 colori dello schermo ST.

L'Atari ST può produrre effetti sonori e pezzi musicali, anche a due o a tre voci. Nel Capitolo 5, *Suoni e musica*, imparerete ad usare le istruzioni SOUND e WAVE per sfruttare a vostro piacimento il sofisticato chip acustico del ST.

Nel Capitolo 6, *Grafica avanzata e programmazione di testo*, imparerete ad usare le funzioni VDISYS e GEMSYS che sfruttano alcune delle velocissime routine interne al computer.

Il Capitolo 7 è intitolato *Come usare il drive e la stampante.* Il computer è anche da solo molto divertente da programmare, ma se volete aggiungergli una stampante, un drive o altre periferiche, potete farlo programmando in BASIC ST. Imparerete anche a programmare file ad accesso casuale e sequenziale su disco.

Il Capitolo 8 tratta *Programmi e problemi.* A questo punto vi sarete abbastanza impadroniti del BASIC ST da sfidare voi stessi in alcuni problemi concreti. Dopo aver letto una delle cinque descrizioni di programma, dovreste essere in grado di tentare un programma da voi. Se avrete bisogno di un pò d'aiuto, potrete consultare i miei suggerimenti e se invece vi troverete completamente bloccati, potrete spiarmi da dietro le spalle mentre definisco la struttura e la stesura di un programma. Potrete batterlo al computer man mano. Una volta portato a termine questo capitolo, avrete cinque programmi interessanti da conservare.

Il libro contiene anche alcune preziose appendici.

L'Appendice A, *Parole chiave del BASIC ST*, è il sommario in ordine alfabetico dell'intero vocabolario del BASIC ST. Ogni voce è corredata del riferi-

mento al Capitolo nel quale potete trovare informazioni ulteriori sulla parola chiave.

L'Appendice B, *Messaggi d'errore del BASIC ST*, elenca tutti i messaggi d'errore che il BASIC ST può presentare. Spiega brevemente il significato di ciascun messaggio e suggerisce come risolvere il problema.

L'Appendice C, *Installazione*, tratta velocemente i procedimenti di installazione e messa in opera di tutti i modelli di computer ST.

L'Appendice D, *I codici ASCII*, è una tabella che riporta tutti i caratteri che si possono produrre con i computer ST e tutti i valori ad essi assegnati nell'American Standard Code for Information Interchange (ASCII ovvero Codice Standard Americano per lo Scambio di Informazioni).

Al momento di questa stesura, la Atari ha previsto di potenziare il BASIC ST aggiungendo alcune nuove parole chiave alla versione che accompagnerà i computer ST venduti dopo il Settembre '86. L'Appendice E, *Ulteriori parole chiave*, raccoglie tali aggiunte.

UNA NOTA SULLE CONVENZIONI TIPOGRAFICHE

Questo testo, come la maggior parte dei libri di programmazione di computer, contiene alcune espressioni che possono risultare estranee ai nuovi utenti. Tali espressioni, quali parole chiave del linguaggio BASIC ST, linee di codifica e output del computer, sono costituite largamente da parole del linguaggio comune. Si distinguono le une dalle altre e dal testo ordinario tramite convenzioni tipografiche. Tali convenzioni a prima vista potranno sembrarvi oscure, ma una volta che le avrete imparate, scoprirete che assolvono al loro compito in modo preciso ed efficace.

- Le parole chiave del BASIC ST compaiono sempre in lettere maiuscole: PRINT, FOR, COLOR e così via. Il computer peraltro non richiede che voi le battiate in questa forma.

- Le istruzioni per il computer e i messaggi che esso vi rimanda sono scritti con caratteri **uguali a questi che sto usando**, nella loro versione in neretto, e compariranno in linee a sé stanti:

```
10  REM Questo è un esempio dei caratteri del programma
```

- Molti degli esempi di programma creano "dialoghi" tra il computer e l'utente. In questi esempi, l'output del computer si distingue per i caratteri in grassetto, più carichi dei precedenti.

INPUT "Come ti chiami?";NME$
Come ti chiami? Tim

- Singole lettere, parole o frasi che rappresentano l'input o l'output del computer e che non si trovano in linee a sé stanti, saranno stampate in corsivo: "Batti il nome che vuoi (*per esempio Giochi o altri*)..."

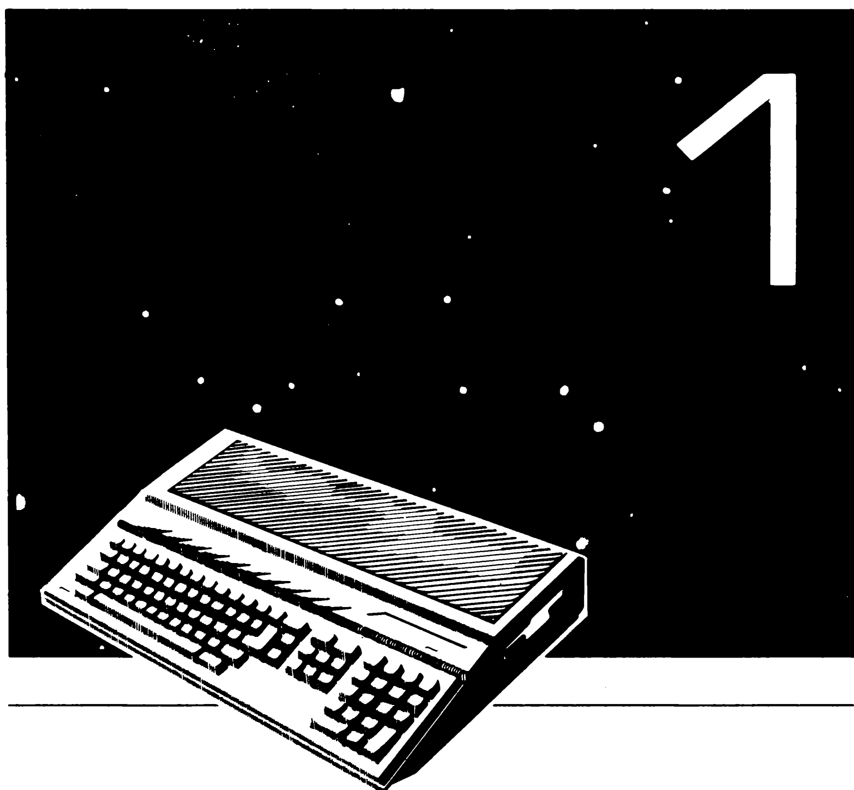
- La notazione più complessa è quella usata per gli esempi di *sintassi (formato).* Questi sono rappresentazioni astratte della forma esatta in cui la parola chiave deve apparire, evidenziando tutte le componenti, sia quelle necessarie che quelle opzionali:

  INPUT ["prompt";]variabile[,variabile, ...]

  I caratteri che fanno fisicamente parte del formato dell'istruzione, la parola INPUT e le virgolette, il punto e virgola e le virgole, sono dello stesso tipo dei normali caratteri del programma. *Il corsivo* viene impiegato per parole in *pseudocodice,* alle quali nelle istruzioni vere e proprie del programma se ne sostituiscono altre, nel nostro esempio il testo del prompt e il nome delle variabili. Le parentesi quadre racchiudono le componenti opzionali, sia vere che in pseudocodice, ed i puntini di sospensione indicano che una componente può essere ripetuta. Nessuna di queste componenti comparirebbe in una istruzione reale, in alcuna forma.

  Quindi potete dedurre che per l'istruzione INPUT un prompt è facoltativo, ma se ne includete uno, lo dovete racchiudere tra virgolette e farlo seguire da punto e virgola. Potete anche vedere che è necessaria almeno una variabile; se ne includete altre, devono essere separate da una virgola.

  In tutto il libro fornirò programmi campione, che potrete battere al computer, usare e sperimentare. Il modo per sfruttare al meglio questo libro è di provare i programmi campione e se seguirete man mano, alla fine della lettura avrete del BASIC ST una comprensione adeguata. Concedetevi al tempo stesso la possibilità di divertirvi; ciò rende l'apprendimento molto più facile e duraturo.



FACCIAMO LA CONOSCENZA DELL'ST

Il 520ST e il 1040 ST Atari sono due nuovi e potenti computer che si distinguono per la grafica a colori, eccellenti capacità di simulazione acustica e per una versione superiore del popolare linguaggio BASIC. Il microprocessore 68000 rende il computer ST veloce e versatile, inoltre la sempre più ricca biblioteca software e le molte periferiche disponibili vi permettono di adattare il sistema alle vostre esigenze. Il 1040ST compare nella Figura 1.1.

Questo libro è una guida completa alla programmazione dell'ST in BASIC. Imparando le parole chiave e le tecniche di programmazione in BASIC ST, potrete far funzionare a vostro piacimento le capacità uniche del 520ST e del 1040ST.

Vediamo un breve elenco dei punti che distinguono i computer ST da altri personal computer:

- Facile operatività con il sistema operativo GEM e il mouse.
- Grafica ad alta risoluzione (fino a 615 per 344 pixel con lo schemo monocromatico e 607 per 166 pixel con lo schermo a colori) e la capacità di visualizzare fino a 16 colori contemporaneamente. Un esempio di grafica ad alta risoluzione è mostrato dalla Figura 1.2.
- Suono e musica, anche fino a tre voci, o canali sonori.
- Una tastiera a 94 tasti, con tasti funzionali, tasti speciali (ad esempio il tasto Help) ed un tastierino numerico, come risulta dalla Figura 1.3.



Figura 1.1: il computer Atari 1040ST.

• 512K di memoria disponibile con il 520ST e 1024K di memoria con il 1040ST (la lettera K è l'abbreviazione di kilobyte; un kilobyte corrisponde a 1024 byte; il byte è l'unità d'informazione più comune del computer).

• Software pronto da usare: il linguaggio di programmazione BASIC, il linguaggio di programmazione LOGO ed il sistema operativo TOS.

• Drive a faccia singola, doppia densità, 360K, oppure a faccia doppia, doppia densità, 720K.

• Una porta MIDI incorporata per i componenti di sintesi del suono ed una porta DMA per un eventuale drive a disco rigido.

• Un prezzo contenuto (al momento di questa stesura circa un milione per il 520ST e un milionecinquecentomila per il 1040ST).

La vera potenza del ST sta nel suo software; le centinaia di titoli di pacchetti disponibili per i computer ST spaziano dai videogiochi ai programmi per ufficio, a quelli per le telecomunicazioni. Per apprezzare il valore del sistema ST, tuttavia, è opportuno avere dimestichezza con tutti i componenti del computer. Leggendo attentamente il prossimo paragrafo, scoprirete l'importanza di ciascun componente del computer, come ciascuno vada collocato nel sistema e come funzionano tutti insieme.



Figura 1.2: un'immagine ad alta risoluzione.

DIFFERENZE TRA I MODELLI

Le differenze tra il 1040ST e il 520ST possono essere esposte molto sinteticamente. La differenza più rilevante è la capacità della memoria, il 1040ST contiene fino a 1024K d'informazioni, mentre il 520ST ne contiene fino a 512K. Il 1040ST inoltre ha un drive incorporato all'unità di tastiera, oltre alla porta per un drive esterno a floppy ed una per un drive a disco rigido. Il 520ST non ha drive incorporati. Per usare un secondo drive con il 520ST dovrete collegare il secondo drive ad una porta del primo. Esistono anche differenze secondarie nella disposizione fisica delle porte e degli interruttori sul retro e sul fianco dell'unità centrale.

Inoltre, sia il 1040ST che i computer 520ST venduti dal febbraio 1986 hanno un chip di ROM (Read Only Memory, ovvero memoria di sola lettura) che contiene il sistema operativo. Per i computer 520ST più vecchi, il sistema operativo veniva fornito su disco, una soluzione leggermente meno pratica. L'Appendice C descrive le procedure di avvio per entrambe le versioni.

I COMPONENTI DELL'ST

Anche se l'ST funziona come un sistema completo, è costituito da centinaia di parti diverse. Per la maggior parte esse vengono raggruppate all'inter-



Figura 1.3: la tastiera dell'ST.

no di alcune unità facilmente individuabili; per esempio, il computer propriamente detto è collocato nella tastiera, che contiene il microprocessore 68000, i chip della RAM (Ramdom-Access Memory, ovvero la memoria ad accesso casuale) e della ROM e le altre parti vitali. Come abbiamo già notato, il computer 1040ST alloggia anche il drive nell'unità tastiera; il 520ST ha invece un drive separato.

La tastiera

L'unità tastiera è in sostanza il computer stesso, poiché contiene il "cervello" (il microprocessore). Per comunicare con il computer, dovete battere lettere, simboli e numeri e fornire altre informazioni che controllano l'operatività del computer, premendo i tasti della tastiera. Alcuni dei 94 tasti forse non vi sono familiari, ma tutti svolgono importanti funzioni. I tasti standard, gli stessi di una qualsiasi macchina per scrivere, sono le lettere dell'alfabeto (A-Z), le cifre (0-9) e vari simboli (!, @, #, $ etc.). La barra è per gli spazi, il tasto Shift vi permette di stampare caratteri maiuscoli o simboli al posto dei numeri ed il tasto Return comunica al vostro Atari che per il momento avete finito di battere. Per esempio, se state scrivendo un programma e volete chiudere la linea di programma corrente, dovrete premere Return; analogamente, se state usando un elaboratore testi per scrivere una lettera e volete chiudere il paragrafo, premerete Return.

Quella che segue è una breve spiegazione.

Tasto Caps Lock

Simile al tasto Shift delle macchine per scrivere, il tasto Caps Lock rende MAIUSCOLE tutte le lettere che batterete. Se battete altri tasti mentre il tasto Caps Lock è premuto, compariranno nel loro formato "maiuscolo" o meglio col carattere più in alto dei due che il tasto riporta. Per disinserire il Caps Lock, basta premere nuovamente il tasto.

Tasto Control

Un carattere di controllo è il risultato della pressione contemporanea del tasto Control e di un'altra lettera. Quindi premendo Control e G contemporaneamente otterrete Control-G. I caratteri di controllo si usano per lo più quando si programma o quando si usa un programma applicativo. Per esempio, useremo Control-C per uscire da un programma BASIC mentre sta girando.

Tasti funzionali

I tasti da F1 a F10 non fanno nulla da soli a meno che non siano programmati per qualche funzione specifica in un programma applicativo. Per esempio, in un programma di telecomunicazioni F1 potrebbe chiamare un certo numero, F7 inviare un messaggio precedentemente memorizzato e F10 interrompere la chiamata.

Tasti freccia

Si usano i tasti freccia in alto, in basso, a destra e a sinistra per spostarsi sullo schermo. Quando avete un testo (lettere e numeri) sullo schermo e volete spostarvi in una certa direzione, basta premere il tasto della freccia corrispondente. Premendo il tasto freccia in alto si sposta il cursore verso l'alto e premendo il tasto freccia a sinistra si sposta verso sinistra. Ogni volta che premete il tasto, il cursore si sposta di uno spazio (o, nel caso dei movimenti verticali, di una linea) nella direzione della freccia.

Tasto Backspace

Si usa questo tasto per tornare indietro di uno spazio e cancellare il carattere che si trova nella posizione del cursore.

Tasto Tab

Il tasto Tab sposta il cursore alla successiva posizione di tabulazione, comunemente fissata di cinque in cinque spazi, per cui premendo Tab tre volte, si sposta il cursore di quindici spazi verso destra. Il tasto Tab è utile per formattare, per esempio per far rientrare il capoverso di un paragrafo quando battete una lettera.

Tasto Help

Alcuni programmi vi permettono d'invocare aiuto nel caso non sappiate più come procedere. Premendo il tasto Help compaiono informazioni utili sullo schermo, se il programma che usate prevede l'utilizzo del tasto Help.

Tasto Undo

Lavorando ad un programma può capitare di fare qualcosa che si preferirebbe non aver fatto. Se il software che state usando riconosce il tasto Undo, potrete tornare indietro e cancellare l'azione più recente. Per esempio, se state usando un programma che prevede l'uso del tasto Undo ed avete accidentalmente cancellato un paragrafo importante, premendo Undo prima di fare qualsiasi altra cosa ricomparirà il paragrafo in questione.

Tasto Alternate

Come il tasto Control, il tasto Alternate svolge in alcuni casi ulteriori funzioni. Non c'è però alcun bisogno di usarlo, salvo che un programma ne richieda esplicitamente l'utilizzo.

Tasto Delete e Insert

I tasti Delete e Insert sono simili, ma svolgono funzioni opposte.

Il tasto Delete cancella il carattere che segue immediatamente il cursore e sposta tutto quello che segue quel carattere di uno spazio verso sinistra, in modo che non restino nel testo spazi vuoti.

Il tasto Insert, al contrario, inserisce uno spazio nel punto in cui si trova il cursore; premendo il tasto Insert un certo numero di volte, si inseriscono nel testo lo stesso numero di spazi nella posizione del cursore. Delete e Insert sono utili quando si sottopone ad editing un programma o quando si apportano cambiamenti ad un testo.

Tasto Clr/Home

Il tasto Clr/Home riporta il cursore nell'angolo in alto a sinistra dello schermo. Questa posizione del cursore si chiama "home" (casa), poiché è per il cursore la naturale posizione di partenza.

Tastierino numerico

Potreste chiedervi perché mai i tasti numerici, diverse funzioni matematiche (+, —, *, /, (), e .) nonché un tasto Enter o Return debbano comparire in due punti diversi: nella tastiera e nel tastierino numerico. Scoprirete molto

presto che il tastierino numerico rappresenta un vero risparmio di tempo quando dovete lavorare su grandi quantità di dati o su problemi matematici. I tasti dei numeri e le funzioni matematiche funzionano esattamente come quelli sulla tastiera ed il tasto Enter del tastierino numerico funziona esattamente come il Return della tastiera principale. Non ci vuole molto tempo per imparare ad usare in modo efficiente il tastierino e sarete presto in grado d'inserire una grande massa di dati numerici in poco tempo.

Il retro della tastiera

Il retro della tastiera non è meno importante della parte frontale. Mentre la parte frontale sarà il vostro mezzo per inserire le informazioni attraverso la tastiera, il retro è il mezzo che il computer impiega per ricevere e inviare informazioni da e per il mondo esterno. Guardate il retro del vostro ST e individuerete la funzione di porte e bottoni. Nei due modelli ST hanno disposizione diversa; i paragrafi seguenti descrivono il retro del 1040ST, che compare in Figura 1.4, procedendo da sinistra verso destra. Se avete invece un 520ST, consultate il vostro manuale.

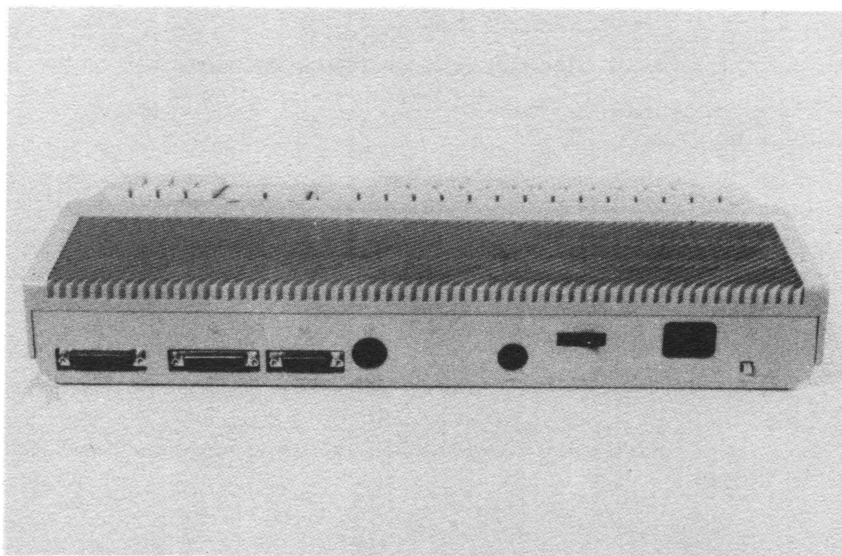


Figura 1.4: il retro della tastiera del 1040ST, che mostra le porte per il computer.

Porta per il modem

Il modem è un dispositivo che permette ad un computer d'inviare e ricevere informazioni attraverso una normale linea telefonica. Se siete interessati a telecomunicazioni e videotex, i modem sono la chiave per accedere in tempo reale a database e ai vari sistemi informativi. Se avete intenzione di prendere un modem, assicuratevi che sia un RS232C-compatibile; dovrete anche guardarvi attorno alla ricerca di software per le telecomunicazioni, se esso non è allegato al modem. L'Emulatore VT52 e la Scrivania GEM ne offrono un po', ma non prevedono tutte le funzioni che normalmente trovereste in un programma specificamente per telecomunicazioni. Anche se molta gente non ha l'effettiva necessità di possedere un modem, in fondo esso costituisce un divertimento e una aggiunta utile ad un sistema. Inoltre, potete usare la porta del modem per collegare una stampante che abbia una porta seriale, nel caso che la vostra stampante non funzioni con una normale porta parallela.

Porta per la stampante

Se pensate che svolgerete elaborazione di testi o che, per qualche altra ragione, avrete bisogno d'informazioni stampate dal vostro ST, utilizzerete la porta per la stampante per collegarne una al computer. Ci sono molti tipi di stampanti, delle quali le più comuni sono quelle a matrice di punti e quelle ad alta qualità di stampa. Le stampanti a matrice di punti sono veloci e relativamente economiche (nell'ordine del mezzo milione, al momento di questa stesura), ma sono più adatte per lettere informali, relazioni o altri documenti che comunque non debbano necessariamente avere l'aspetto di copie dattiloscritte. Le stampanti ad alta qualità di stampa sono più costose e non altrettanto veloci, ma il tipo dei caratteri è lo stesso di una macchina da scrivere. La scelta dovrà dipendere dalle vostre esigenze e dalla somma che siete disposti a spendere, comunque l'aggiunta di una stampante rende un computer molto più utile.

Porta per disco rigido

Se il vostro lavoro richiede un drive più veloce e più capiente del normale drive per dischetti, potreste prendere in considerazione il drive a disco rigido. In questa ipotesi, lo dovrete collegare alla porta a disco rigido. Un disco rigido può accogliere fino a 25 volte il numero di informazioni (10 o 20 megabyte) in più rispetto ad un dischetto e ad una velocità nettamente superiore rispetto ad un normale drive. L'Atari offre un drive da 20 megabyte e potete

comunque svolgere un'indagine sui drive proposti da altri produttori. Questa porta è conforme al nuovo standard SCSI (Small Computer System Interface, ovvero interfaccia per piccoli sistemi computerizzati), che permette un accesso veloce ai drive a disco rigido e ad altre periferiche.

Porta per dischetti

Il drive che accompagna il sistema 520ST va collegato alla porta per dischetti. Il 1040ST, naturalmente, non richiede porta per il suo primo drive, che è incorporato, ma dispone di una porta per un secondo drive. Se decideste di collegare un secondo drive al 520ST, basterebbe collegare un cavo dalla porta sinistra sul retro del primo drive alla porta destra del secondo.

Porta per lo schemo

La porta per lo schermo collega il vostro schermo monocromatico o a colori al computer e permette al computer d'inviare i suoi segnali video allo schermo ad alta risoluzione.

Interruttore On/Off

L'interruttore On/Off, ovviamente, accende e spegne il vostro ST. State attenti a questo bottone: una volta che il computer sia spento, la memoria del ST viene completamente cancellata.

Porta di alimentazione (Power)

Questa è la porta attraverso la quale il sistema attinge energia elettrica. La spina circolare del cavo di alimentazione si inserisce in questa porta. Accertatevi che il sistema sia spento quando effettuate il collegamento.

Bottone Reset

Il bottone Reset riavvia l'ST come se lo doveste spegnere e riaccendere. Siccome spegnere e riaccendere non è un bel modo di riavviare il computer (esiste il rischio, per quanto piccolo, di danneggiare l'hardware), usate il bottone Reset tutte le volte che il computer si blocca in un programma senza lasciarvi

altra via d'uscita o quando volete riavviare il computer per cancellarne la memoria. State attenti con il bottone Reset: premendolo, la memoria viene totalmente ripulita.

Altre porte

Cartucce

Sul lato sinistro sia del 520ST che del 1040ST c'è una feritoia dove vanno inserite le cartucce con chip da 128K di ROM. Quando questa cartucce vengono inserite, potete istantaneamente usare programmi di dimensioni fino a 128K. Dato che l'intero programma è memorizzato nel chip della ROM, non occupa la memoria del computer e si carica istantaneamente, poiché il collegamento a cartuccia offre al computer un accesso immediato al chip di ROM. Le cartucce sono un mezzo efficiente per caricare i programmi, anche se la maggior parte dei programmi su cartuccia sono giochi.

Porta 0 e porta 1

I computer ST hanno anche altre due porte. La porta 0 di solito è riservata al mouse e si può usare la porta 1 per il joystick. Se volete usare due joystick, potete temporaneamente staccare il mouse e collegare il secondo joystick; basta che non vi dimentichiate di ricollegare poi il mouse, poiché è una parte importante del vostro ST.

Porte MIDI

Il MIDI, sigla per Musical Instrument Digital Interface (ovvero interfaccia digitale per strumenti musicali) è uno standard usato dai musicisti per gli strumenti musicali elettronici, come i sintetizzatori a tastiera. Con il MIDI, potete collegare strumenti musicali sofisticati al vostro computer. L'ST serve come stazione di programmazione per lo strumento. Poiché l'ST è MIDI-compatibile, potreste risparmiare un sacco di soldi se prevedete di lavorare con strumenti elettronici sofisticati, poiché non dovrete comprare un altro computer per farli funzionare. La porta MIDI di accesso permette all'ST di ricevere un input dal sintetizzatore o da altro strumento e la porta MIDI di uscita invia informazioni (come volume, velocità e così via) allo strumento. Potrete comporre musica elaborando una traccia per volta con il 520ST o il 1040ST, usando il computer in connessione con strumenti musicali dotati di MIDI.

Periferiche

Schermi

Esistono due diversi tipi di schermo (altrimenti detti video o monitor) a disposizione dei computer ST. Quello monocromatico, SM124, è uno schermo verde ad alta risoluzione che può visualizzare testo e grafici con una risoluzione fino a 640 per 400 pixel, o punti. Il monitor a colori, SC1224, può visualizzare fino a 512 colori diversi (di cui 16 contemporaneamente sullo schermo nella modalità 320 per 200 pixel), con una risoluzione che raggiunge i 640 per 200 pixel.

Drive

Il drive è un importante dispositivo input/output. Potete memorizzare programmi e informazioni su qualsiasi dischetto da 3 1/2 pollici che inseriate nel drive e potrete rileggere nel computer le stesse informazioni in un secondo momento. Il drive può memorizzare le informazioni in modo permanente; il computer perde qualsiasi dato dalla memoria quando lo si spegne o qualora si verifichino dei problemi d'irregolarità dell'alimentazione (ad esempio un picco o una momentanea riduzione di tensione). Trattate con attenzione sia

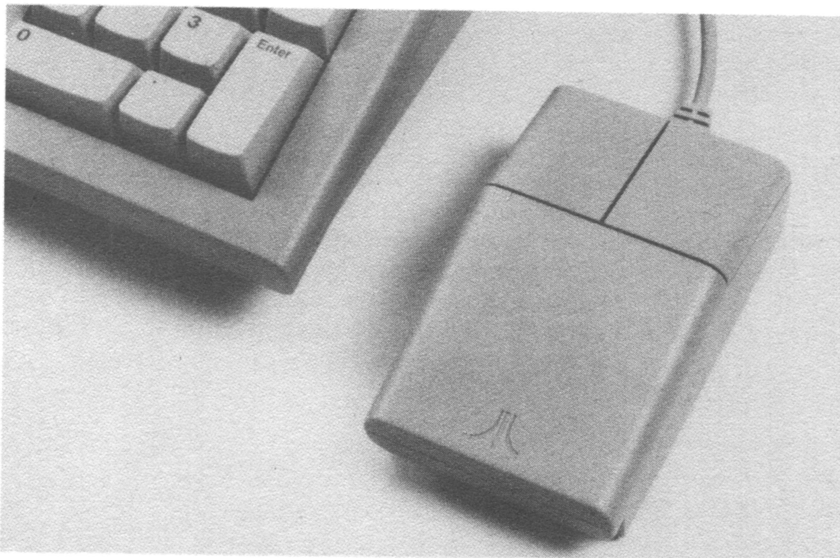


Figura 1.5: il mouse, un importante dispositivo di input.

il drive che i singoli dischetti e ricordatevi sempre di fare una copia di tutti i dischetti importanti (vedi la discussione sulla scelta del formato nel menù File, più avanti in questo capitolo, per imparare a produrre delle copie).

Mouse

Il mouse è un dispositivo d'input importante in un sistema ST, perché vi permette di spostare un puntatore per tutto lo schermo per fornire comandi al computer. Per esempio, immaginate di dover puntare ad un certo programma che volete usare, oppure d'indicare usando il mouse un particolare paragrafo che volete cancellare da una lettera. Il mouse ha due bottoni, ma userete quasi sempre quello a sinistra. Spostando il mouse su una superficie piana, spostate il puntatore (o qualche altro simbolo corrispondente) per lo schermo e premendo il bottone di sinistra, potete selezionare le voci che compaiono sullo schermo. Il mouse è presentato nella Figura 1.5.

LA SCRIVANIA GEM

La scrivania GEM (Graphics Environment Manager) costituisce il vostro collegamento con il sistema operativo, noto come TOS (sistema operativo Tramiel). Un sistema operativo è controllore e gestore delle risorse del computer, permettendo l'interazione tra la CPU (Central Processing Unit, ovvero unità centrale di elaborazione) e le periferiche, i dischi e la memoria. Potete usare la scrivania GEM per spostare i file da un disco ad un altro, fare copie di un disco, accedere ai file e svolgere altre operazioni relative ai dischi e all'uso che il computer ne fa. Molti computer, come il PC IBM, impiegano un sistema operativo nel quale si battono tutti i comandi direttamente dalla tastiera. In quel caso, se voleste copiare un file chiamato PROGRAM.BAS dal drive A al drive B, dovreste battere qualcosa come:

```
COPY A: PROGRAM.BAS B:
```

Invece il GEM visualizza tutto nella forma di piccoli disegni chiamati icone. Le icone dall'aspetto di schedari rappresentano i dischetti. L'icona che si presenta come un cestino per rifiuti è il posto dove i file vengono "buttati via". Altri tipi di programma e di file hanno icone diverse. Per gli utenti esperti, il mouse costituisce un dispositivo d'input efficiente e potente quando viene usato nell'ambito del sistema GEM.

La scrivania GEM, diversamente dal BASIC ST, non è affatto un linguag-

gio. Si tratta invece di un programma specializzato che vi aiuta a gestire le infomazioni che si trovano sui vostri dischi. Benché dobbiate trovarvi nella scrivania GEM per accedere al BASIC ST, il GEM e il linguaggio BASIC si escludono reciprocamente; non hanno niente a che fare l'uno con l'altro, tranne per il fatto che basta un solo comando per passare dall'uno all'altro.

Per usare la scrivania si ricorre al mouse. Provate a muovere il mouse in tutte le direzioni, in movimenti circolari o a caso. Potrete notare che la freccia che compare sullo schermo si sposta nella stessa direzione del mouse. Per **aprire** l'icona del dischetto e vedere quali file contiene, muovete il mouse finché la freccia si trova sull'icona del dischetto A. Non basta che punti in quella direzione: la punta della freccia deve effettivamente trovarsi sovrapposta all'icona. Una volta posizionato il puntatore, premete due volte il bottone sinistro del mouse, senza indugiare troppo tra un click e l'altro. In questo modo l'icona diventa scura (ovvero viene **evidenziata**), indicando che è stata selezionata, prima che la finestra si apra. Qualsiasi oggetto deve essere selezionato in questo modo, prima che altre operazioni possano essere eseguite su di esso.

La finestra del disco

Quando si apre l'icona di un disco, ne comparirà la finestra. La finestra, che contiene le icone di tutti i programmi e i file di quel disco, è raffigurata nella Figura 1.6 ed è costituita dalle parti di cui segue la descrizione:

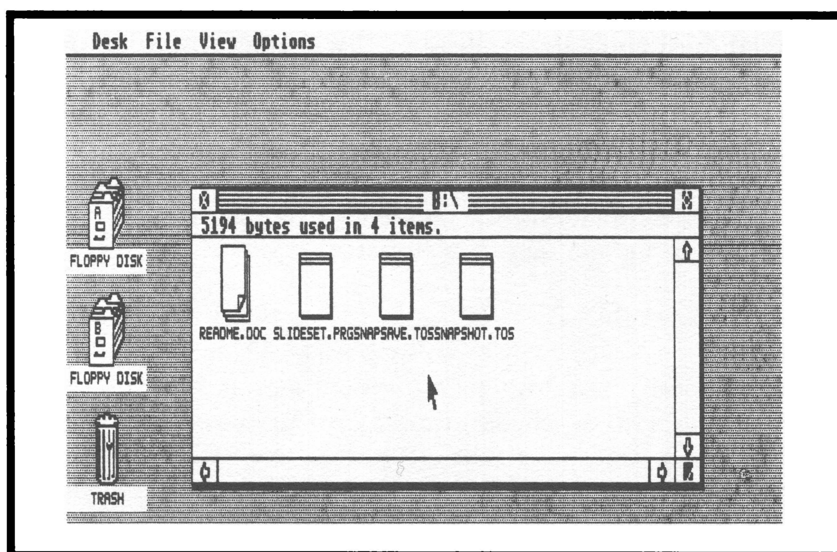


Figura 1.6: la finestra relativa ad un dischetto.

Casella di chiusura Nell'angolo in alto a sinistra della finestra c'è una casella, che può chiudere la finestra. Se spostate la freccia su questa casella e premete il bottone sinistro del mouse, la finestra si richiude, in modo contrario a come si è aperta.

Barra di spostamento La parte più alta della finestra è la barra di spostamento, che ha l'aspetto di un fascio di righe grigie orizzontali. Vi permette di spostare la finestra per tutto lo schermo. Portate il puntatore sulla barra, tenete premuto il bottone del mouse e potrete così spostare il profilo della finestra. Quando l'avrete portato nel punto che volete, lasciate andare il bottone e la finestra andrà ad occupare il profilo.

Identificatore di disco e linea d'informazione La finestra dell'identificatore si trova al centro della barra di spostamento e riporta la directory (lista dei nomi di file contenuti) attuale. Nel nostro caso, riporterà **A:** poiché state usando il disco nel drive A. La linea d'informazione, situata appena al di sotto dell'identificatore e della barra di spostamento, dice quanti byte del disco sono stati occupati e quanti file si trovano nella directory corrente. Tenete presente che non tutte le icone devono essere necessariamente visualizzate nella finestra; se la finestra è troppo piccola per visualizzarle tutte, dovrete ricorrere alla freccia di scorrimento (che vedremo tra poco) per spostare la finestra in modo da poter vedere il resto delle icone.

Casella d'ingrandimento Nell'angolo in alto a destra della finestra c'è la casella d'ingrandimento. Quando la puntate e premete il bottone del mouse, essa fa in modo che la finestra del disco vada ad occupare tutto lo schermo. Se poi vorrete ridurre la finestra alle sue dimensioni originali, potrete premere il bottone un'altra volta sulla casella d'ingrandimento, che si trova nella stessa posizione anche sulla finestra ingrandita.

Casella di dimensionamento Se volete modificare le dimensioni della finestra, o se volete vederne più di una per volta, spostate il puntatore nell'angolo in basso a destra della finestra e, mentre tenete premuto il bottone sinistro del mouse, spostando la casella cambierete le dimensioni della finestra. Quando le proporzioni del profilo corrispondono alle dimensioni da voi desiderate, lasciate andare il bottone e la finestra assumerà le nuove dimensioni. Cambiare le dimensioni può rivelarsi utile quando volete vedere tutte le icone e la finestra corrente non è abbastanza grande da mostrarle tutte.

Frecce di scorrimento Come abbiamo già accennato, si può far scorrere il contenuto di una finestra a destra, sinistra, in alto e in basso per vedere le icone che a causa delle dimensioni della finestra rimangono nascoste. Per far-

la scorrere orizzontalmente, dovrete selezionare Show as Icons nel menù View. Non si può invece far scorrere un testo. Se volete far scorrere un'icona, portatevi sulla freccia che indica la direzione in cui la volete muovere, poi tenete premuto il bottone sinistro del mouse tutto il tempo necessario a percorrere la distanza da voi stabilita. Le due caselle di scorrimento, situate l'una tra la freccia in alto e la freccia in basso e l'altra tra la freccia a destra e la freccia a sinistra, indicano la distanza rimanente entro la quale potete scorrere la finestra. Dovete tenere premuto il bottone del mouse e poi lasciarlo per ogni icona che volete far scorrere.

Come si usa la finestra

Ora che conoscete le diverse parti di una finestra, vediamo alcuni impieghi che possiamo fare della finestra stessa nell'ambito della scrivania GEM (la scrivania compare nella Figura 1.7). Tenete presente che se volete spostare una icona, dovete innanzitutto portare il puntatore direttamente sull'icona. Poi, tenendo premuto il bottone sinistro, spostate il mouse in modo da portare (o **trascinare**) l'icona ovunque vogliate. Una volta posizionata l'icona, potete lasciar andare il bottone del mouse. Ogni volta che selezionate un'icona o la spostate sopra l'icona del cestino o del disco, tale icona verrà evidenziata; se puntate all'icona di un programma e premete il bottone del mouse, essa verrà

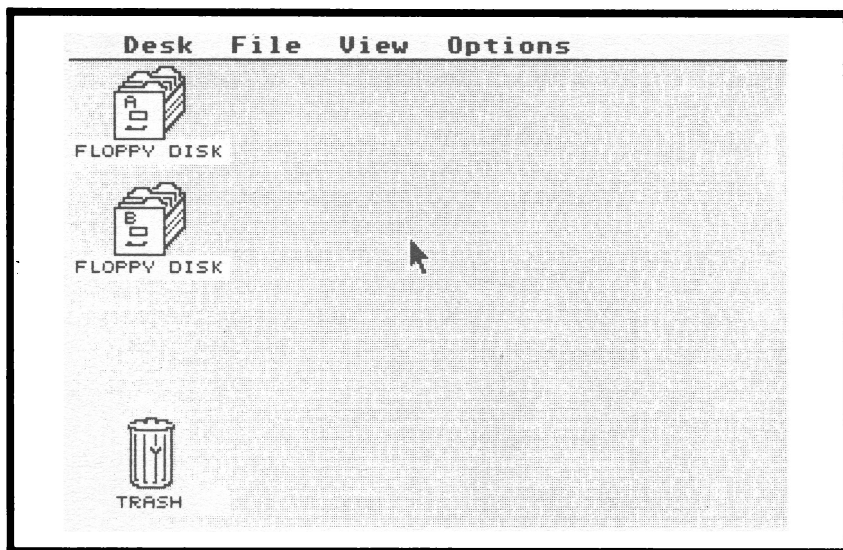


Figura 1.7: la scrivania GEM.

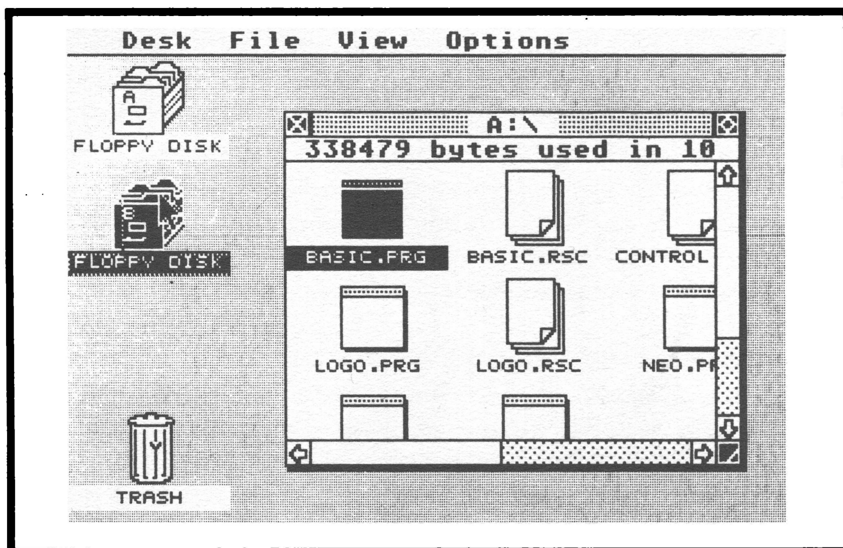


Figura 1.8: l'icona di un dischetto viene evidenziata quando portate il file su quel disco.

evidenziata (indicando che è stata selezionata). Se portate un'icona sopra il cestino per buttarla via, l'icona che state spostando verrà evidenziata ed il cestino verrà a sua volta evidenziato quando gli sarete vicini abbastanza da permettere al computer di capire che intendete buttar via qualcosa. Ecco alcuni modi di usare la finestra.

Copiare un file Se volete copiare un programma o un file da un disco ad un altro, innanzitutto assicuratevi che ci sia un dischetto in ogni drive (o se avete un sistema con un solo drive, accertatevi che la finestra con il file in questione sia aperta sullo schermo, poi inserite nel drive il disco sul quale volete riprodurre la copia). Poi, spostate l'icona del file dalla finestra in cui si trova al momento, sull'icona del disco dove volete destinare la copia. Per esempio, se avete un sistema a due drive, spostate l'icona del file sull'icona del disco; l'icona del disco B (nell'ipotesi che vogliate copiare sul disco che si trova nel drive B) verrà evidenziata quando l'icona del file si troverà "in" quel disco. La Figura 1.8 mostra le icone evidenziate. Quando lasciate andare il bottone del mouse, il programma farà una copia del file sul drive B.

Eliminare un file Se volete togliere in modo definitivo un file da un disco, portate l'icona dalla finestra al cestino, come nella Figura 1.9. Come abbiamo già detto, il cestino verrà evidenziato. Quando lascerete il bottone, il computer farà comparire una casella di dialogo sullo schermo se l'opzione Set Pre-

ferences è predisposta in tal senso. Le caselle di questo tipo compaiono in molte diverse applicazioni del ST e non fanno altro che richiedervi una risposta; in questo caso l'Atari vuole conferma che vogliate effettivamente cancellare una voce o che non preferiate invece cancellare il comando, perchè una volta che l'avrete buttata via, sarà praticamente impossibile recuperarla. Se volete veramente buttar via la voce in questione, premete il bottone sulla casella dell'OK e avrete così cancellato la voce in modo permanente; se avrete invece premuto il bottone con il puntatore nella casella di cancellazione, il comando verrà cancellato e ritornerete alla solita scrivania GEM.

Usare un file Se volete usare un programma o "aprire" un file, basta eseguire lo stesso procedimento che abbiamo visto per aprire l'icona del disco: basta premere due volte sulle icone con le quali volete lavorare. Per esempio, c'è un programma sul disco Language che si chiama SAMPLE.PRG. Portate il puntatore sulla relativa icona e premete due volte il bottone sinistro del mouse (anche qui, non fate alcuna pausa tra un click e l'altro). Vi troverete allora nel programma; questo è un programma particolarmente interessante, perché vi insegna il funzionamento delle finestre, facendovele usare direttamente. La finestra sullo schermo contiene un ovale colorato, e potrete cambiare questa finestra come qualsiasi altra. Provate la casella di dimensionamento e la barra di spostamento; noterete che quando cambiate le dimensioni della casella o spostate la finestra, anche il colore dell'ovale nella finestra cambia. Quando

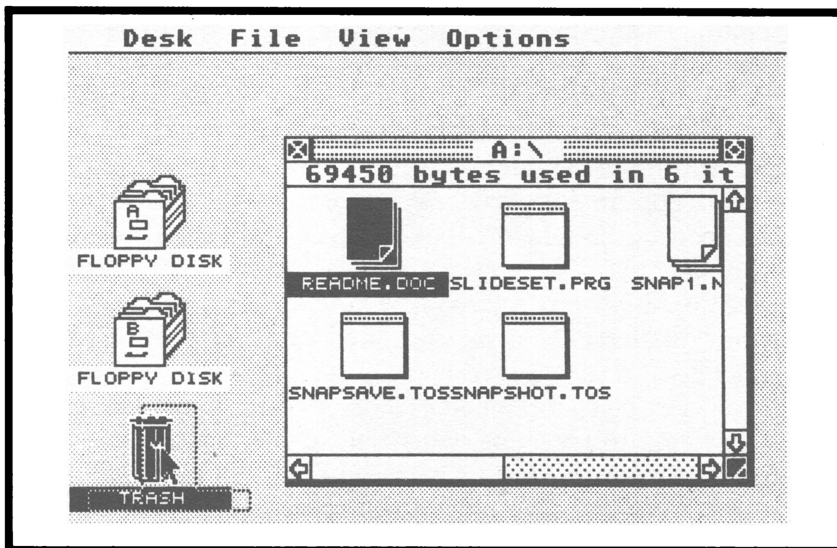


Figura 1.9: portate il file sull'icona del cestino per buttarlo via.

avete finito di giocare con la finestra del SAMPLE.PRG, puntate la casella di chiusura per ritornare alla scrivania GEM.

Scegliere più di un'icona Se volete copiare diversi file da un disco ad un altro, non è necessario che spostiate un'icona per volta, aspettando che ciascun file sia copiato ad uno ad uno. Potete selezionare diversi file contemporaneamente racchiudendoli in un rettangolo e spostandoli in blocco. Per selezionare più di un'icona, portate il puntatore in un punto esterno alle icone, rimanendo però nell'angolo in alto a sinistra rispetto all'area che volete circoscrivere; tenendo premuto il bottone sinistro del mouse, iniziate a spostarvi in basso a destra e noterete che, man mano che spostate il mouse, si forma un rettangolo punteggiato.

Quando vi trovate nell'angolo in basso a destra dell'area che comprende tutte le icone che volete selezionare, lasciate andare il bottone sinistro del mouse. A questo punto, verranno selezionate tutte le icone che si trovano all'interno della cornice punteggiata, come nella Figura 1.10 e potrete ora lavorare sul gruppo nel suo insieme, anziché sulle singole. Potrete buttarle via, copiarle su un altro disco o farne qualsiasi altra cosa.

Scoprirete presto che lavorare con le finestre e le icone può essere di gran lunga più facile che avere a che fare con comandi di testo. Ora vediamo un altro elemento importante della scrivania GEM di cui dovete venire a conoscenza: la barra del menù.

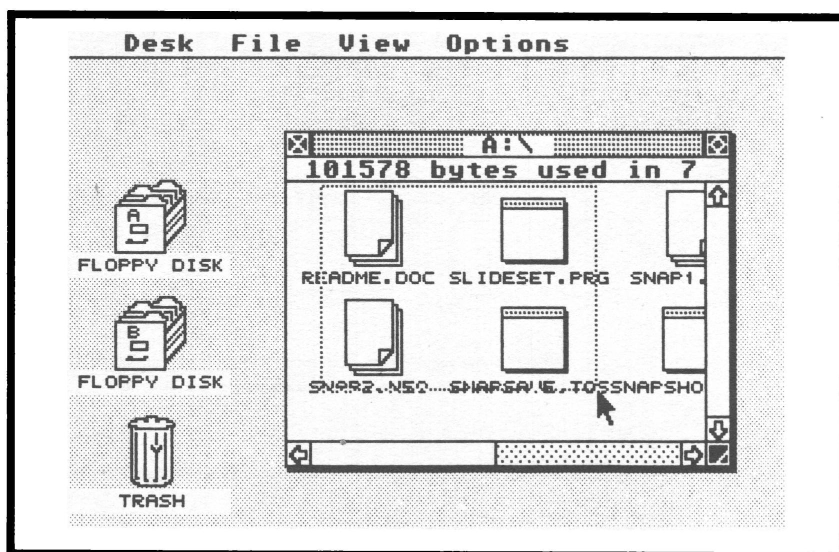


Figura 1.10: la cornice punteggiata mostra quali file avete selezionato.

La barra dei menù

Nella parte alta dello schermo della scrivania GEM vedrete la **barra dei menù**, contrassegnata da queste parole:

Desk File View Options

La barra dei menù è un insieme di comandi che semplificano l'uso della scrivania GEM. Quando vi portate con il mouse su una di queste voci, comparirà un menù appena al di sotto della voce selezionata, rivelando diverse opzioni, come nella Figura 1.11. Per selezionare una di queste opzioni, basta muovere il puntatore su quella che volete e poi premere due volte il bottone sinistro del mouse.

Se spostando il puntatore in prossimità della barra dei menù, ne selezionate uno per errore, basta allontanare il puntatore e premere il bottone una sola volta. Ciò farà sparire il menù e vi permetterà di riutilizzare normalmente il mouse.

La maggior parte delle voci nascoste nei menù sono piuttosto importanti, quindi le analizzeremo una per volta sotto i quattro titoli dei rispettivi menù.

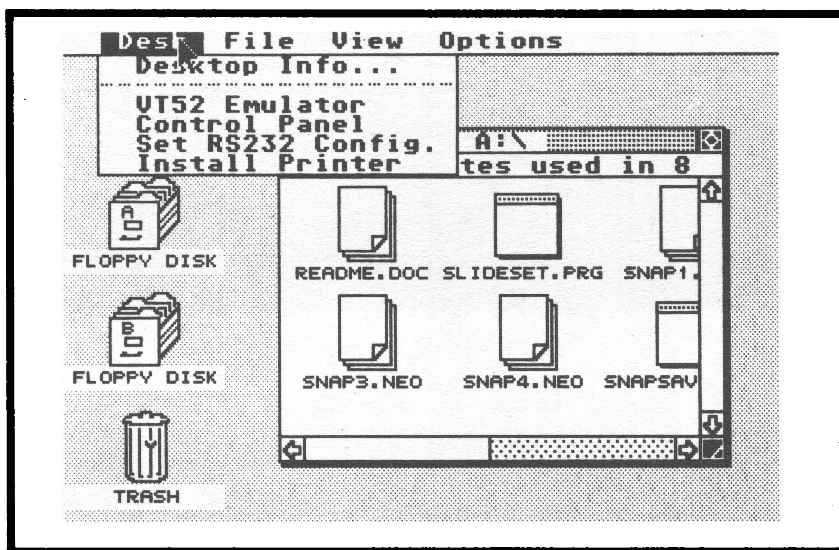


Figura 1.11: selezionando una delle voci della barra dei menù si apre un nuovo menù.

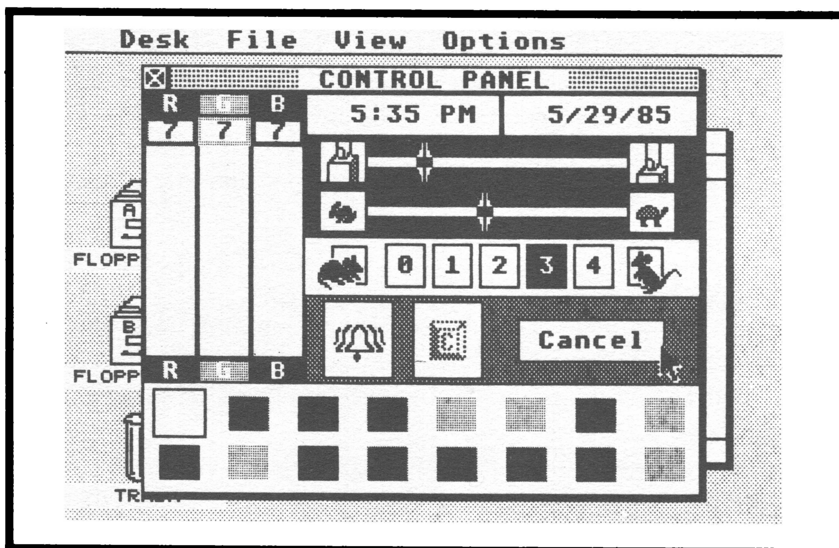


Figura 1.12: il pannello di controllo.

Menù Desk

Desktop Info Se volete scoprire il nome completo e le note di copyright del programma che state usando, premete due volte puntando su Desktop Info. Nel caso del disco del sistema, l'Atari compone un rettangolo sullo schermo contenente le informazioni relative al programma:

GEM, Graphics Environment Manager (c) 1985 Digital Research Corp.

Sullo schermo ci sono anche altre informazioni, e una volta che le avrete lette, puntate la casella dell'OK per ritornare alla scrivania GEM.

VT52 Emulator Se avete un modem o qualche altra interfaccia collegata alla porta del modem, il VT52 Emulator è un semplice programma in GEM che vi permette di comunicare con altri computer. Una volta che abbiate selezionato il VT52 Emulator, basta premere il tasto Help per impostare se necessario la 4 configurazione del terminale all'altra estremità. Per comunicare con altri computer, basta cominciare a scrivere. Qualsiasi cosa battiate verrà inviata alla porta del modem, in modo tale che lo stesso testo comparirà sul terminale all'altro capo della linea di telecomunicazioni. Se comperate un modem, preferirete probabilmente disporre di software più sofisticato di questo, ma il VT52 è un complemento utile al GEM. Per ritornare alla scrivania GEM, premete il tasto Undo.

Control Panel Il Pannello di Controllo, come nella Figura 1.12, è un'opzione importante poiché vi permette di regolare il vostro ST. Potete regolare l'orologio o il calendario, nonché la sensibilità della tastiera e dei bottoni del mouse, i messaggi sonori emessi dal computer mentre battete e la tavolozza di colori che intendete impiegare. Il pannello di controllo è una finestra esso stesso, poiché potete modificare ognuna delle voci suddette usando il mouse e la tastiera.

Clock/Calendar La regolazione dell'orologio e del calendario è localizzata nella parte alta del pannello di controllo. Per cambiare l'ora, puntate la parte sinistra della barra dove si trova l'ora; quando la casella dell'ora sarà evidenziata, battete alla tastiera l'ora corretta nel formato riportato dal pannello di controllo (per esempio **08:53 AM**).

Per aggiornare il calendario, puntate la rispettiva casella ed inserite dalla tastiera la data esatta, anche in questo caso nel formato suggerito (per esempio, **05/12/86**). Finché il computer rimane acceso, l'ora e la data corrette rimarranno in memoria.

Tempo di risposta della tastiera Ci sono due controlli regolabili in modo continuo posti appena al di sotto delle caselle dell'ora e della data. Il controllo più in alto mostra sulla sinistra un dito che preme solo leggermente un tasto e a destra un dito che lo preme a fondo; il controllo in basso riporta un coniglio a sinistra e una tartaruga a destra. Il controllo più in alto dice al computer dopo quanto un carattere dovrà iniziare a ripetersi, quando il rispettivo tasto sia stato premuto. Se spostate il controllo verso sinistra, non ci vorrà molto dopo aver premuto un tasto perché il carattere cominci a ripetersi. Spostandolo verso destra, direte al computer di attendere un pò di più prima di ripetere il carattere. Potete fissare il controllo in un punto qualsiasi tra i due estremi. Lo stesso vale anche per il controllo in basso, che stabilisce la velocità con cui il computer dovrà far succedere i caratteri ripetuti. Muovendo il controllo verso il coniglio i caratteri si ripetono rapidamente (per esempio, premendo il tasto A per un istante, potrebbe risultare sullo schermo AAAAAAA), spostandolo invece verso la tartaruga, il carattere si ripeterà più lentamente.

Tempo di risposta dei bottoni del mouse È già capitato di dire diverse volte che non dovete indugiare troppo tra una pressione e l'altra del bottone del mouse. Potrete voi stessi stabilire l'intervallo tra una pressione e l'altra perché il ST continui ad interpretarla come un doppia pressione. Se di solito premete il bottone del mouse velocemente, potrete fissare questo controllo vicino al topolino in piedi; se tendete a lasciare un po' più di tempo tra un click e l'altro del mouse, dovrete fissarlo più vicino al topolino sdraiato. Potete fissare il

tempo di risposta del mouse a 0, 1, 2, 3 o 4, con lo 0 dalla parte del topolino sdraiato e il 4 verso il topolino in piedi.

Audio Feedback Ad alcune persone piace sentire un suono tutte le volte che premono un tasto sulla tastiera, mentre altre trovano la cosa fastidiosa. Ed ancora, alcune persone vogliono sentire suonare un campanello tutte le volte che commettono un errore, altre lo trovano imbarazzante o irritante. Potete inserire o togliere le due caratteristiche selezionando la casella corrispondente. La casella con il disegno di una campanella controlla il segnale acustico dell'errore; la casella sulla destra, che riporta il tasto C, controlla il feedback acustico della tastiera. Potrete **inserirle o disinserirle** semplicemente selezionandole con il mouse. Quando le icone sono scure la funzione è attivata, l'icona vuota indica invece che la funzione è disattivata.

Palette Controls Se avete un monitor a colori, potete cambiare ognuno dei 16 colori in uso sullo schermo usando i controlli della tavolozza. I 16 quadratini colorati nella parte bassa del pannello di controllo riportano i 16 colori residenti in memoria. Potete puntare ad uno qualsiasi di essi per cambiare quel particolare colore. Potrete cambiare tutti i colori che vorrete, uno per uno. Una volta che avete selezionato il colore che volete cambiare, potete regolare i tre controlli continui sul lato sinistro del pannello di controllo. Essi controllano l'intensità dei livelli di rosso, di verde e di blu nel colore che avete selezionato e muovendoli potrete ottenere il colore che desiderate. Dato che si può

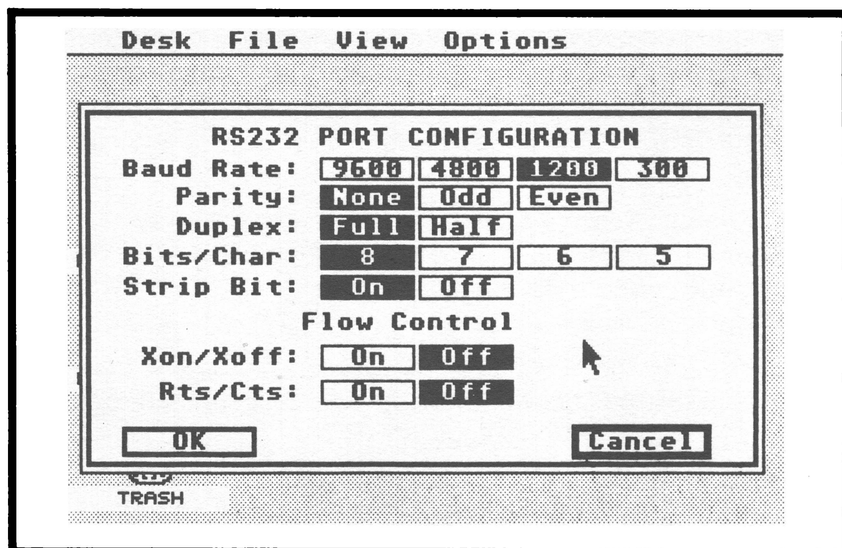


Figura 1.13: il menù Set RS232 Configuration.

regolare ogni controllo con notevole precisione, avete a disposizione ben 512 colori tra cui scegliere.

Quando avrete finito di usare il pannello di controllo, potete chiudere la finestra puntando la sua casella di chiusura, poi potete salvare su disco quello che avete scelto, usando l'opzione **Save Desktop** nel menù Options.

Set RS232 Configuration Se state usando la porta del modem, potete conformare la porta al vostro modello di modem ricorrendo all'opzione Set RS232 Configuration. RS232 è semplicemente una porta seriale per computer che devono inviare e ricevere infomazioni. La Figura 1.13 riporta il menù di questa opzione.

Dovrete impostare le seguenti voci.

Baud Rate Il tasso di Baud è la velocità alla quale il vostro modem invia le informazioni via cavo telefonico, misurate in bit al secondo. Potete scegliere tra 9600, 4800, 1200 o 300 baud. La velocità minima e più comune per gli home computer è 300 baud.

Parity Se lavorate su un computer che verifica che le informazioni che state inviando siano ricevute in modo chiaro e corretto, potete fissare la parità, dispari o pari, secondo quella richiesta dal computer ricevente. Potrete altrimenti non fissare alcuna parità.

Duplex Se siete in linea con un computer che vi rinvia tutto ciò che gli comunicate, dovrete usare la modalità full-duplex. Se il computer ricevente si limita a ricevere le vostre informazioni senza rifletterle indietro, dovrete porvi nella modalità half-duplex, in modo da poter vedere quello che state battendo.

Bit/Char, Strip Bit, Xon/Xoff, Rts/Cts Potrete regolare le opzioni Bit/Char, Strip Bit, Xon/Xoff e Rts/Cts in base al particolare computer con il quale state comunicando. Potrete conservare quasi sempre le loro attuali impostazioni.

Install Printer L'opzione Install Printer è importante solo se avete aggiunto una stampante al vostro sistema. Dovrete specificare le seguenti caratteristiche.

Printer Type Per questa opzione potrete selezionare Dot per stampanti a matrice di punti e Daisy per stampanti ad alta qualità di stampa.

Color Le scelte di colore possono essere B/W (bianco e nero) per la maggior parte delle stampanti e Color per le stampanti a colori.

Pixel/Line L'opzione Pixel/Line determina se la stampante ha 1200 o 960 pixel per riga.

Quality Le scelte sulla qualità di stampa sono Draft (brutta copia) per ottenere copie più veloci e di qualità più povera e Final per copie più lente, a più alta qualità.

Printer Port L'opzione Printer Port vi permette di scegliere la porta Printer per una stampante ad interfaccia parallela oppure la porta Modem per una stampante ad interfaccia seriale.

Paper Type L'opzione Paper Type permette di comunicare al computer se state usando un normale modulo continuo (Feed) oppure se deve fermarsi per permettervi di inserire nella stampante un foglio per volta (Single).

Menù File

Open Probabilmente non userete l'opzione Open molto spesso, perché premere due volte il bottone sinistro del mouse è più veloce e più facile. Tuttavia se volete aprire l'icona di un disco con l'opzione Open, puntate al programma o al file che volete aprire, premete una volta il bottone sinistro del mouse, poi selezionate Open dal menù File.

Show Info Quando avete selezionato un'icona, potete reperire informazioni relative a quel programma, file o disco selezionando l'opzione Show Info. Potete scoprire informazioni quali il nome del file, quanti byte occupa e quanti byte sono rimasti liberi sul disco. Questo è anche il vostro mezzo per cambiare il nome dei file, ma non quello di un disco. Quando selezionate Show Info e puntate ad un'icona, potete usare il tasto Backspace per cancellare il nome del file e poi battere il nome nuovo. Una volta che avrete premuto Return o puntato alla casella dell'OK, uscirete dalla opzione Show Info e all'icona verrà assegnato il nome nuovo. Potete anche usare questa opzione per avere una breve descrizione del cestino.

New Folder Un folder (cartella) serve per raccogliere diversi documenti raggruppandoli per argomento. Immaginate i vostri problemi come se fossero dei fogli di carta separati che potreste mettere in una certa cartella. Se i fogli di carta sono sparsi, li potete vedere tutti, ma non sono organizzati. Se li riordinate in diverse cartelle, non potrete vederli se non aprendo la relativa cartella. Potrete archiviare i singoli programmi "all'interno" di certi folder. Quando avrete ordinato i vostri documenti in una cartella, non li potrete consultare

se non aprendola (premendo due volte il bottone sinistro del mouse, come al solito). Una volta che il folder è aperto, potrete vedere tutti i file ed usare quelli che volete. Potrete avere più di un folder aperto per volta. L'opzione New Folder crea l'icona di una cartella, alla quale potete assegnare un nome. Quando selezionate New Folder, non dovete far altro che battere il nome e puntare la casella dell'OK. Per ritornare alla scrivania GEM, puntate la casella di cancellazione. La Figura 1.14 mostra l'inserimento del nome di una nuova cartella.

Close e Close Window Le opzioni Close e Close Window svolgono funzioni simili. L'opzione Close dà lo stesso risultato che puntare la casella di cancellazione sulla finestra: chiude cioè la finestra in uso al momento. Anche in questo caso, è poco probabile che userete spesso l'opzione Close. L'opzione Close Window è un pò più completa, poiché chiude tutti i folder del disco, oltre a chiudere la finestra. Quando una finestra o un folder sono chiusi, non potete più accedere ai loro file; quando sono aperti, le icone sono visibili e potete quindi selezionarle.

Format Quando volete usare un disco nuovo (per fare una copia di riserva di materiale importante o per memorizzare alcune delle vostre creazioni), dovrete formattare il disco in modo che possa lavorare con il drive dell'ST. Un dischetto vuoto da 3 1/2 pollici si adatta al PC portatile IBM, al Macintosh della Apple, all'ST Atari e ad altri computer, ma nessuno di questi computer

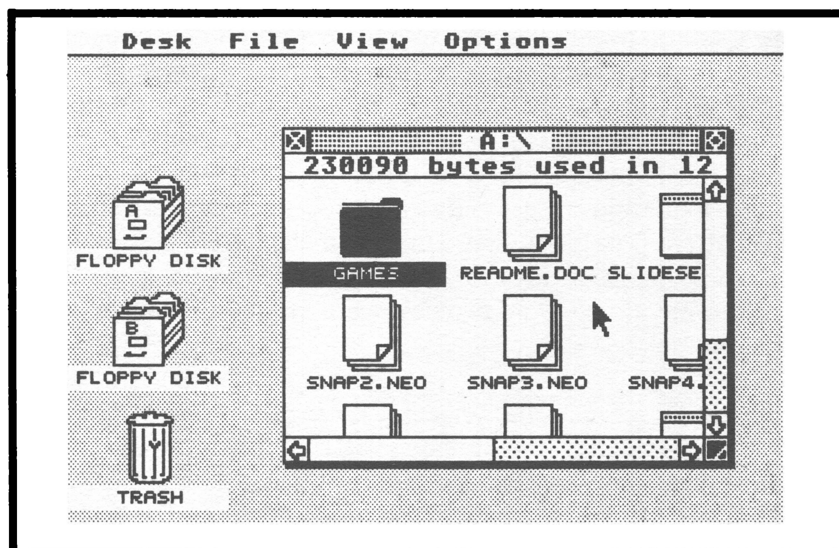


Figura 1.14: una cartella chiamata Games.

può leggere i dischi formattati dagli altri, quindi è necessario formattarlo specificamente per gli ST. Dopo aver scelto Format, l'ST vi avverte che formattando il disco cancellerete ogni eventuale informazione che esso contenga. Potete a questo punto scegliere OK per procedere o Cancel per uscire dall'operazione di formattazione. Poi dovete battere il nome che volete dare al disco, lungo fino ad otto caratteri. Scegliete un nome che richiami il contenuto generale del disco, ad esempio **Giochi.** Potete anche decidere se usare entrambi i lati del disco (se avete drive e dischi a faccia doppia) oppure se ne userete uno solo. Quando avrete selezionato l'opzione Format, l'ST inizierà a formattare il disco, operazione che richiede solo pochi istanti. L'ST vi dirà poi di quanti byte di memoria potete disporre sul disco.

Making a Backup Innanzitutto è opportuno che conosciate un ulteriore passaggio richiesto per riprodurre una copia di un disco. Formattare un disco vuoto è solo il primo passo. Una volta fatto questo, aprite la finestra del disco di cui volete eseguire la copia. Se avete solo un drive, estraete il disco dal drive ed inseritevi il disco vuoto; se invece avete due drive, inserite il disco vuoto nel drive B. Infine, portate l'icona del disco A sull'icona del disco B, come risulta dalla Figura 1.15. Sullo schermo comparirà un messaggio che dice che state per copiare un disco e che vi chiede di confermare che è proprio ciò che intendete fare. Dopo la vostra conferma, l'ST copierà il contenuto del primo disco sul disco vuoto (per i possessori di un solo drive, l'ST darà man mano

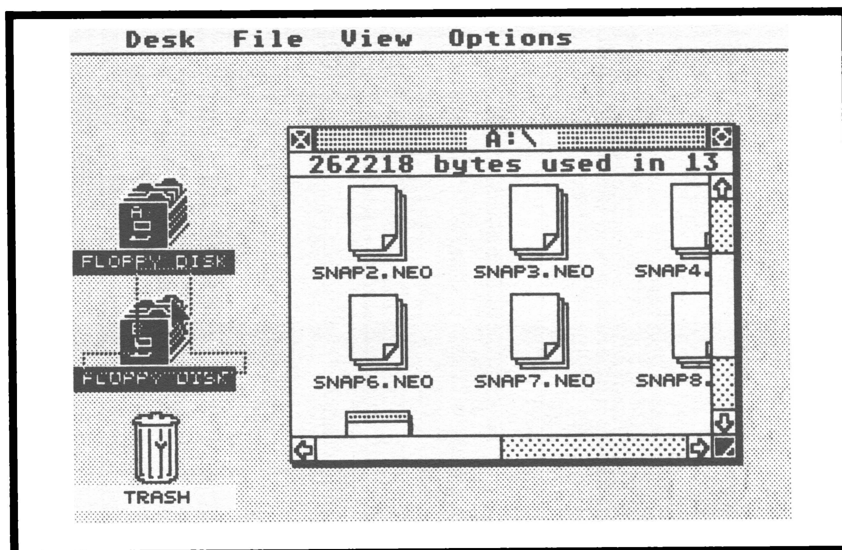


Figura 1.15: l'icona del dischetto A portata sull'icona del dischetto B.

le istruzioni circa il disco da inserire nel drive durante le varie fasi della copiatura; basterà che seguiate le istruzioni).

Menù View

Il menù View ha solo una funzione: vi permette di specificare in quale forma volete che i programmi e i file del disco vengano visualizzati sulla scrivania GEM. Potrete scegliere tra le opzioni seguenti.

Show as Icons L'impostazione di default del computer è Show as Icons. Questa opzione visualizza le diverse voci di un disco nella forma d'**icone**, ovvero piccole immagini, anziché come testo.

Show as Text Se preferite leggere il testo che le figure, selezionate l'opzione Show as Text. Il computer visualizzerà i nomi e le dimensioni dei file, insieme all'ora e alla data in cui sono stato modificati per l'ultima volta. La Figura 1.16 mostra lo schermo di questa opzione.

Si usano le linee di testo esattamente come le icone, dato che le potete spostare, usarle a gruppi, o eseguire su di esse qualsiasi altra funzione che abbiamo visto relativamente alle icone.

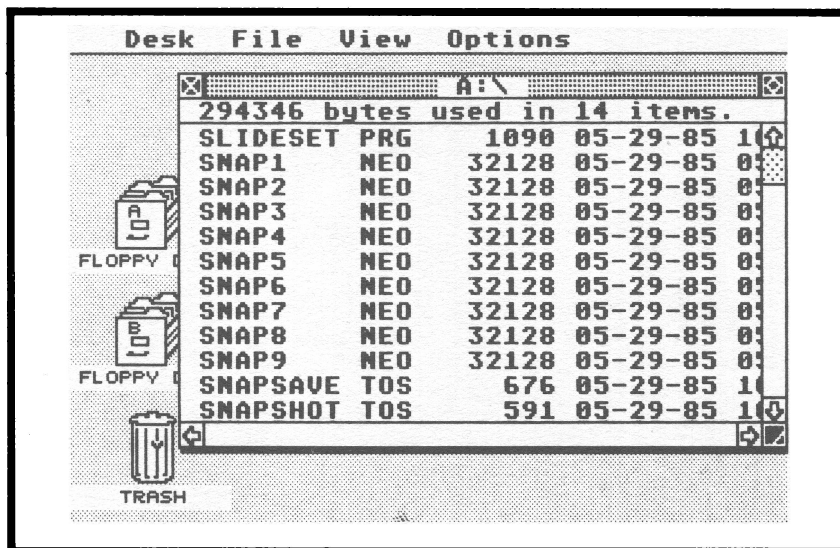


Figura 1.16: file visualizzati come testo.

Sort by Name L'opzione Sort by Name visualizza i file in ordine alfabetico. Vengono elencati per primi i folder, ordinati per nome, seguiti dagli altri file, anch'essi ordinati per nome.

Sort by Date L'opzione Sort by Date visualizza i file ordinati in base alla data in cui sono stati modificati per l'ultima volta, iniziando dal file più recentemente modificato.

Sort by Size L'opzione Sort by Size visualizza i file ordinati in base alle loro dimensioni, misurate in byte, dal più grande al più piccolo.

Sort by Type L'opzione Sort by Type pone i file in gruppi di programmi, file di dati, o qualsiasi altro tipo di file che abbiate su disco. Li ordina in base al suffisso di tre caratteri che denota il tipo di file.

Menù Options

Install Disk Drive Se volete cambiare i nomi alle icone dei dischi, togliere un'icona, oppure aggiungerne un'altra, usate l'opzione Install Disk Drive. Quando selezionate quest'opzione, comparirà una casella che vi permetterà di cambiare l'etichetta sull'icona del drive. Per esempio, potreste cambiare drive **A** in **System Disk.** Qualsiasi nome scegliate, dovete puntare la casella di installazione per apportare il cambiamento, la casella Remove per eliminare l'icona del drive o la casella di cancellazione per uscire dal comando.

Install Application L'opzione Install Application è una funzione complessa usata di solito da programmatori esperti. Essenzialmente, vi permette di specificare se il GEM verrà usato con un programma particolare e quale tipo di file di dati aprirà un'**applicazione**, che non è altro che un programma usabile. Questa funzione trascende gli scopi di questo libro (se volete saperne di più, consultate il manuale del ST Atari)

Set Preferences Ci sono tre componenti dell'opzione Set Preferences. Le prime due, Confirm Deletes e Confirm Copies, dice al computer se deve chiedervi conferma quando cancella o copia un file. Se siete utenti provetti, può darsi che non vogliate aver a che fare con una o entrambe queste conferme. Tuttavia pensateci bene se volete impostarle sul No, perché è sempre utile che il computer esegua un doppio controllo per verificare che non stiate commettendo errori. La terza componente, Set Screen Resolution, vi permette di specificare la **risoluzione** o la finezza della grafica (nonché eventualmente il numero di colori a vostra disposizione). Se il vostro schermo è monocromatico, dovete usare High Resolution (alta risoluzione). Se avete uno schermo a colo-

ri, potete selezionare la risoluzione Low (bassa) o Medium (media). La risoluzione Low vi permette di avere tutti i 16 colori, ma dà lettere grosse e grafica a bassa risoluzione. La risoluzione Medium permette solo quattro colori, ma le lettere sono più adatte all'elaborazione testi e ad applicazioni commerciali e la grafica è ad alta risoluzione.

Save Desktop Quando cambiate l'aspetto della scrivania - come con l'opzione Set Preferences o con il Pannello di Controllo - potete rendere permanenti questi cambiamenti. Per farlo, selezionate l'opzione Save Desktop; altrimenti i vostri cambiamenti rimarranno in atto solo finché non spegnerete il computer.

Print Screen Se avete una stampante grafica, potete stampare l'immagine sullo schermo con il comando Print Screen. Verrà incluso nell'immagine anche il puntatore del mouse e prima preferirete forse portarlo fuori dall'immagine.

Tasti dei movimenti del puntatore

Se, per qualche ragione, dovete spostare il puntatore usando la tastiera anziché il mouse - per esempio se non l'avete e se non funziona - potrete usare in sostituzione i seguenti tasti che controllano il puntatore, per quanto in modo più disagevole.

Alternate-Freccia Premete il tasto Alternate contemporaneamente ad un tasto freccia per spostare il puntatore di otto pixel nella direzione della freccia.

Alternate-Shift-Freccia Premete Alternate, Shift ed un tasto freccia contemporaneamente per spostare il puntatore di un pixel nella direzione della freccia.

Alternate-Insert Premete i tasti Alternate e Insert contemporaneamente per selezionare (e quindi evidenziare) l'icona alla quale il puntatore sta puntando.

Alternate-Clr/Home Premete contemporaneamente i tasti Alternate e Clr/Home per svolgere la stessa funzione di premere il bottone destro del mouse.

Alternate-Insert-Freccia Premete contemporaneamente Alternate, Insert e un tasto freccia per spostare l'icona selezionata nella direzione della freccia.

Tipi d'icone

Prima di entrare nel BASIC ST, diamo un'ultima occhiata ai diversi tipi di icona che possiamo incontrare sulla scrivania GEM.

Icona di disco

I dischi A e B sono rappresentati dalle icone di disco, situate sul lato sinistro della scrivania GEM. Potete cambiare il nome di queste icone con l'opzione Install Disk Drive sulla barra dei menù. Quando aprite un'icona di disco, ne viene visualizzato il contenuto.

Icona del cestino dei rifiuti

I file che non servono più finiscono nel cestino dei rifiuti. Quando cercate di buttar via qualcosa, il programma vi chiederà conferma dell'operazione (se Set Preferences è impostato in tal senso), ma state lo stesso attenti quando volete buttare le icone nel cestino.

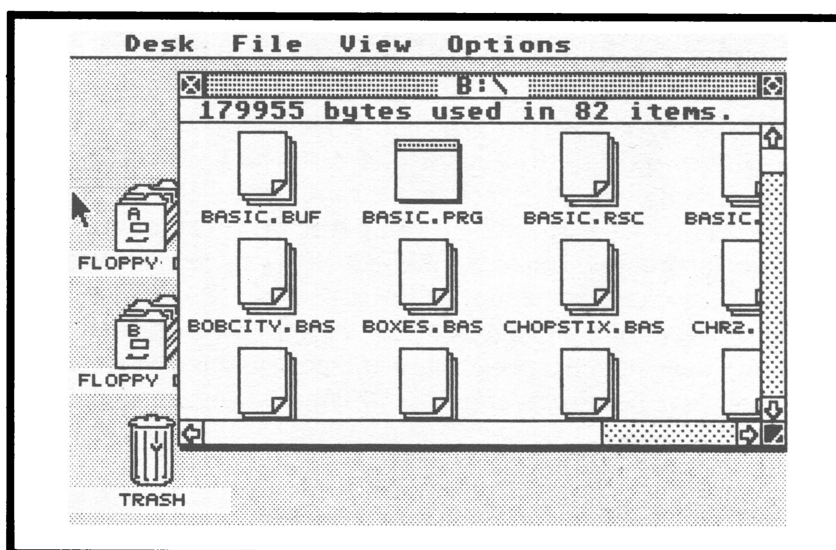


Figura 1.17: un'icona non di programma.

Icona di programma

L'icona di un programma si presenta come un quadratino il cui bordo superiore riporta una linea punteggiata. I programmi, come i programmi BASIC che scriverete più avanti, assumono questa icona. Prendono la forma di un'icona di programma anche quelli in linguaggio più avanzato di tipo assemblativo, nonché molte altre applicazioni. Per caricare e lanciare un programma rappresentato dall'icona, basta selezionare quell'icona.

Icone non di programma

Molti file, come documenti di testo, immagini grafiche e istruzioni di programma non sono programmi. Come nella Figura 1.17, questi file sono rappresentati come una pila di fogli di carta con l'angolo in basso a destra della pagina superiore ripiegato.

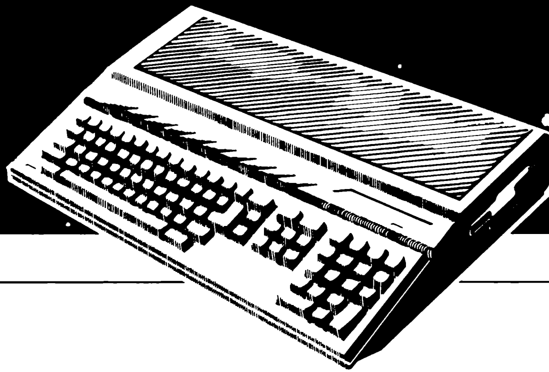
Icona del folder

L'icona del folder è facile da individuare, perché ha l'aspetto di una cartella. Come abbiamo già avuto occasione di dire, si usano i folder per ordinare altri file e si creano con l'opzione New Folder dalla barra dei menù.

ENTRIAMO NEL BASIC ST

Ora che ho trattato gli aspetti principali della scrivania GEM, mi concentrerò sull'argomento del libro: il BASIC ST. Estraete il disco che si trova nel drive A e inserite il disco ST Language (che contiene il BASIC ST). Selezionate l'icona del disco A e dovrebbero comparire i file memorizzati su questo disco. Portate il puntatore sull'icona BASIC.PRG, premete due volte il bottone sinistro del mouse e preparatevi a lavorare con il BASIC ST.

Nel prossimo capitolo, imparerete alcuni concetti fondamentali del BASIC e presenterò alcuni semplici programmi che costruiranno le fondamenta di cui potrete servirvi per procedere verso programmi più impegnativi.



ELEMENTI FONDAMENTALI DEL BASIC

Il linguaggio di programmazione BASIC non è difficile da imparare, ma prima di poter tracciare grafici o di scrivere programmi, è necessario che impariate alcuni elementi fondamentali del BASIC e alcune regole di programmazione. Poi, sarà molto più facile imparare le parole del linguaggio BASIC (le *parole chiave*) e la loro funzione.

IL LINGUAGGIO BASIC

BASIC è la sigla di Beginner's All-purpose Symbolic Instruction Code, ovvero linguaggio di Istruzioni simboliche per Principianti. Per quanto il BASIC sia un ottimo linguaggio per coloro che non hanno familiarità con i computer (come suggerisce la definizione stessa), rimane tuttavia un linguaggio versatile e potente, ottimo per programmatori principianti e di livello intermedio. La versione Atari del linguaggio, il BASIC ST, è particolarmente adatta ai computer 520ST e 1040ST, specialmente per quanto riguarda la grafica.

Una delle ragioni per cui è importante imparare il BASIC è che vi permette di conoscere più a fondo le potenzialità del vostro computer e di sfruttarle a vostro vantaggio. Ma forse l'aspetto più importante è che diventare provetti conoscitori di un linguaggio per computer vi abitua a pensare in modo più chiaro e veloce. Poiché un computer per funzionare richiede istruzioni esatte e precise, presto scoprirete che anche il vostro modo di pensare diventa più preciso ed analitico. Ma non abbiate paura di cominciare a "pensare come un computer": i computer non pensano affatto, hanno l'intelletto e l'immaginazione di un interruttore.

Si usa il BASIC per dire al computer cosa volete che faccia. Per dire al computer quali sono i compiti che deve assolvere, dovrete essere esatti e precisi. Quindi dovrete battere le istruzioni, chiamate *programma*, linea per linea, in modo ordinato.

Tipi di parole chiave del BASIC

Le parole chiave che formano il vocabolario del BASIC ST si possono raggruppare in tre categorie: *istruzioni, comandi* e *funzioni.* Questi termini hanno un significato preciso nel contesto della programmazione dei computer e l'intento di questo libro è di usarli in modo coerente. Un'istruzione fa parte di un programma in BASIC e dice al computer di compiere una data azione. Diversamente dalle funzioni, le istruzioni sono autonome, cioè non è necessario che facciano parte di un'istruzione più ampia per dare il via ad una cer-

ta azione nell'ambito del programma. I comandi sono anch'essi autonomi. Differiscono dalle istruzioni per il fatto che non possono entrare a far parte di un programma. Al contrario essi vanno impiegati in maniera *immediata.* Le funzioni sono istruzioni che *restituiscono* i valori che risultano dallo svolgimento di certe operazioni. Esse istruiscono il computer in modo che svolga determinate operazioni, ma non possono farlo in modo indipendente; esse devono far parte di un'istruzione nell'ambito di un programma. Capirete meglio il significato di queste distinzioni quando studierete dettagliatamente le parole chiave.

LE FINESTRE DEL BASIC ST

Il BASIC ST vi aiuta a scrivere i programmi mettendovi a disposizione quattro diverse finestre con cui lavorare: Command, List, Output e Edit. Come potete vedere nella Figura 2.1, quando entrate nel BASIC ST la finestra Edit rimane nascosta dietro le altre tre. Queste finestre funzionano sostanzialmente come le finestre della scrivania GEM, poiché anch'esse hanno una casella di ingrandimento, una casella di chiusura, una casella di dimensionamento, una barra di spostamento e una barra per i titoli. Vediamo una breve descrizione di ciascuna.

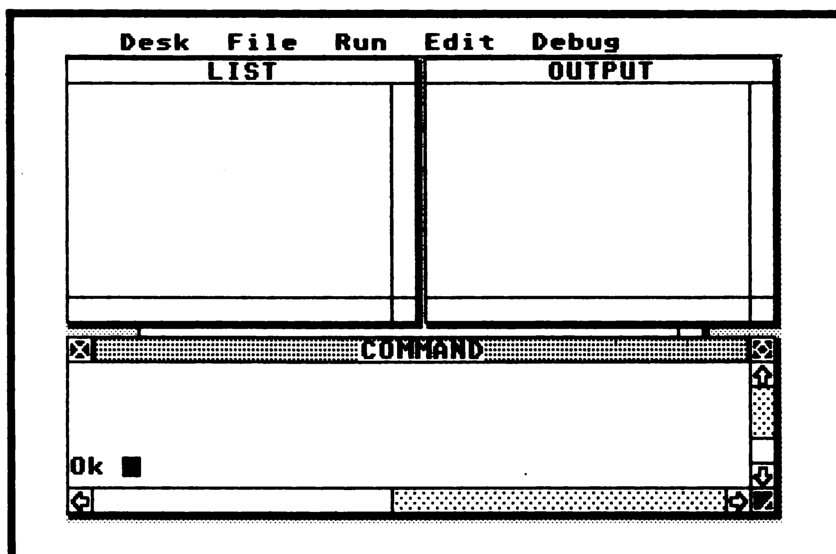


Figura 2.1: le finestre del BASIC ST.

La finestra Command

La finestra Command è il luogo in cui si trova il cursore quando entrate nel BASIC ST. La finestra Command vi permette d'inserire sia comandi diretti, sia linee di programma. Quando state programmando, potete usare questa finestra per aggiungere linee di programma o per compiere operazioni speciali (come trovare il valore di una variabile, cancellare certe linee e salvare un programma su disco). La finestra Command è quella che userete più spesso, dato che è il vostro mezzo per far sapere al computer cosa volete che faccia o quali linee vanno inserite nel programma che state creando.

La finestra List

Quando inserite il comando List dalla finestra Command, la finestra List visualizzerà un *listato*, cioè il contenuto del programma. Dovrete probabilmente far scorrere parecchio la finestra per poter vedere tutto il listato del programma su cui state lavorando, ma la finestra List costituisce comunque un modo veloce per esaminare il programma con tutte le linee riordinate. Più avanti vedremo che i numeri di linea dicono al computer in quale punto una certa linea va collocata rispetto al resto del programma. La finestra List mostra le linee di programma in ordine sequenziale.

La finestra Output

Quando *lanciate*, od *eseguite*, un programma, l'output verrà visualizzato nella finestra Output. Ci sono dei comandi del BASIC ST che vi permettono di allargare qualsiasi finestra all'intero schermo; troverete comodo dilatare la finestra Output all'intero schermo quando un programma sta girando, dato che in tale situazione le altre tre finestre non servono.

La finestra Edit

La finestra Edit rimane nascosta dietro le altre tre finestre. La si può vedere, tuttavia, attivandola esattamente come qualsiasi altra finestra della scrivania GEM o del BASIC ST, vale a dire portando la punta del puntatore del mouse direttamente sulla finestra e premendo il bottone sinistro del mouse. Poiché della finestra Edit compare solo una piccola striscia tra le finestre List, Output e Command, può darsi che dobbiate fare successivi tentativi prima che compaia la finestra Edit. Essa funziona esattamente come la finestra List,

dato che mostra tutte le linee di un programma. La differenza è che quando vi trovate nella finestra Edit, potete apportare cambiamenti al contenuto di un programma. Potete aggiungere o cancellare linee al programma, cambiare il contenuto di una certa linea o eliminare i *bugs*, cioè gli errori di programmazione. Esaminerò tutte la funzioni di editing nell'ultimo paragrafo di questo capitolo.

Come si usano le finestre del BASIC ST

Per scoprire come funzionano le finestre Command e Output, battete al computer la linea che segue, premendo Return alla fine di essa:

PRINT "Sono quassu', nella finestra Output!"

Dopo aver premuto Return (cosa che dovrete sempre fare per concludere qualsiasi comando o linea di programma), la frase racchiusa tra virgolette dovrebbe comparire nella finestra Output, come nella Figura 2.2. Avete inserito un comando diretto ed il computer l'ha eseguito riportando la frase nella finestra Output.

Con i comandi diretti, non si ricorre mai alle finestre List o Edit. Esaminerò l'editing più avanti, ma per vedere come funziona la finestra List, inserite

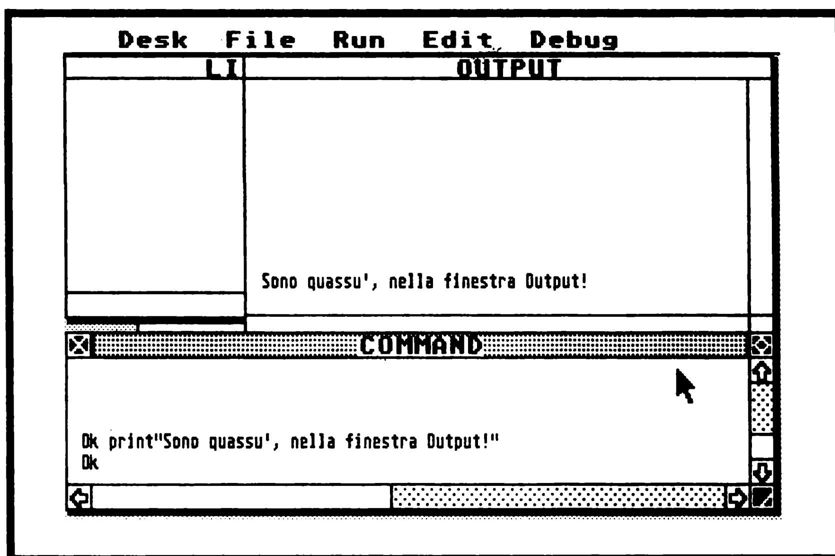


Figura 2.2: il computer stampa la frase nella finestra Output.

il breve programma che segue. Non preoccupatevi ora di capire cosa significhino le linee; limitatevi a batterle al computer esattamente come sono (premete Backspace se battete un carattere sbagliato e lo volete cancellare) e premete Return alla fine di ogni linea. Il computer saprà che non ci sono comandi diretti, perché sono precedute da un numero di linea.

```
10 A=1
20 Print "La variabile numerica A e' uguale a ";A
30 A=A+1
40 GOTO 20
```

Ora battete *LIST* e premete Return; il nostro programma dovrebbe comparire nella finestra List. Se avete difficoltà a vederlo tutto, espandete la finestra in modo che ci stia tutto il programma. Ricordate che LIST visualizza l'intero programma che avete battuto al computer. Un altro comando importante è RUN, che dice al computer di avviare il programma; battete *RUN* e premete Return per avviare il nostro programmino.

La finestra Output presto conterrà frasi che riportano valori crescenti di A, come segue:

La variabile numerica A e' uguale a 1
La variabile numerica A e' uguale a 2
La variabile numerica A e' uguale a 3
(e così via)

Quando sarete stanchi di vedere A diventare sempre più grande, tenete premuto il tasto Control mentre premete il tasto C; questo comando che si definisce Control-C, dice al computer d'interrompere il programma e di tornare alla finestra Command.

Se volete salvare un programma in BASIC su disco, usate il comando SAVE, seguito dal nome del file tra virgolette. Il nome del file può comporsi di otto caratteri al massimo ed è opportuno che richiami lo scopo o il contenuto del programma. Per esempio, se scrivete un programma per disegnare sullo schermo dei fiori, potreste salvarlo su disco battendo:

```
SAVE "FIORISTA"
```

Potete controllare quali file ci sono sul disco battendo *DIR* (per *directory*), seguito dal tasto Return; imparerete meglio l'uso del comando DIR nel paragrafo sui comandi relativi al disco nel Capitolo 7. Infine, quando volete caricare un programma dal disco, basta battere *LOAD*, seguito dal nome del file (anche in questo caso racchiuso tra virgolette). Per riavere il programma FIO-

RISTA, non dovrete far altro che battere *LOAD "FIORISTA"*.

L'ultimo semplice comando diretto da imparare a questo punto è NEW; batterete molti esempi di programma nel corso di questo libro, ma l'ST non può sapere quando avete concluso di lavorare con un programma e volete usarne un altro. NEW dice al computer di cancellare il programma che ha attualmente in memoria; tutte le volte che volete smettere di usare un certo programma, battete *NEW* e premete Return prima d'iniziare con il successivo. Battete *NEW* adesso per cambiare il programma attualmente in memoria.

Dovete anche sapere che la maggior parte dei comandi diretti possono anche essere selezionati da menù, oltre che battuti alla tastiera. LIST si trova nel menù Edit; RUN è nel menù Run; SAVE è nel menù Save as File; LOAD è nel menù Load. Inoltre selezionando Break o Stop dal menù Run, si ottiene lo stesso effetto che battendo Control-C.

La barra dei menù del BASIC ST

Noterete che lo schermo del BASIC ST ha una propria barra dei menù nella parte superiore (vedi Figura 2.3) che riporta i seguenti titoli:

Desk File Run Edit Debug

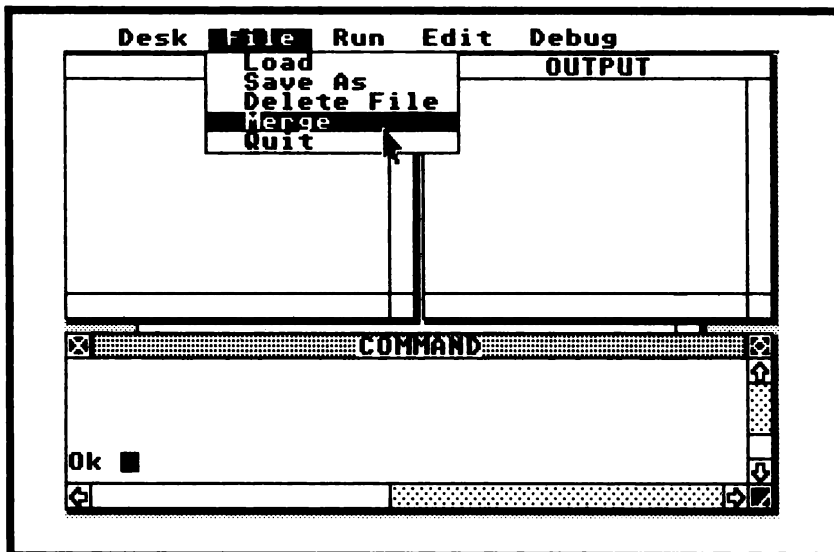


Figura 2.3: la barra dei menù del BASIC ST.

I menù hanno le seguenti opzioni. Tra parentesi ho indicato i comandi o le parole chiave equivalenti da usare dalla finestra Command per ottenere gli stessi risultati.

MENÙ DESK

About ST BASIC Informazioni relative al Copyright del BASIC ST.

VT52 Emulator È identico a quello della scrivania GEM.

Control Panel È identico a quello della scrivania GEM.

Set RS232 Configurations È identico a quello della scrivania GEM.

Install Printer È identico a quello della scrivania GEM.

MENÙ FILE

Load L'opzione Load carica un programma in BASIC dal disco (dalla finestra Command, battete LOAD seguito dal nome del file in questione tra virgolette; per esempio, LOAD "PROGMIO").

Save As L'opzione Save As salva su disco il programma che c'è in memoria, sotto il nome che specificherete (equivalente: SAVE).

Delete File L'opzione Delete File cancella un programma dal disco (equivalente: ERA).

Merge L'opzione Merge carica un programma dal disco e lo aggiunge al programma attualmente in memoria (il comando MERGE fa la stessa cosa, ma dovreste stare sempre attenti che i numeri di linea del programma in memoria e quelli da caricare dal disco siano diversi, altrimenti le linee del programma che state caricando vanno a sostituirsi, cancellandole, a quelle con lo stesso numero di linea del programma già in memoria).

Quit L'opzione Quit vi permette di uscire dal BASIC ST e di tornare alla scrivania GEM (equivalenti: QUIT o SYSTEM).

MENÙ RUN

Run L'opzione Run lancia il programma (equivalente: RUN).

Break L'opzione Break interrompe il programma che sta attualmente girando (equivalente: STOP).

Stop L'opzione Stop fa andare il programma nella modalità di riposo nella quale potrete battere STOP per fermare il programma o continuarlo.

Continue L'opzione Continue riprende il programma da dove è stato interrotto dopo aver incontrato un comando BREAK o un'opzione Stop (equivalente: CONT).

Step L'opzione Step fa girare il programma un passo per volta, interponendo una pausa tra una linea e la successiva. Questo è utile nella fase di editing, quando volete controllare il risultato di ciascuna linea del programma (equivalente: STEP).

Buf Graphics L'opzione Buf Graphics vi permette di memorizzare i grafici che sono sullo schermo, in modo che possiate richiamarli in un secondo tempo.

MENÙ EDIT

Start Edit L'opzione Start Edit vi permette di porvi con la finestra Edit nella modalità di editing (equivalente: EDIT).

Exit Edit L'opzione Exit Edit vi permette di uscire dalla modalità di editing e ritornare alla finestra Command.

Help Edit L'opzione Help Edit mostra i comandi che i tasti funzionali svolgono nella modalità di editing.

Goto Line L'opzione Goto Line vi permette di andare ad un certo numero di linea nella modalità di editing (quando selezionate quest'opzione, il computer vi chiede il numero di linea).

Delete Lines L'opzione Delete Lines elimina le linee che specificate (equivalente: DELETE; il computer vi chiederà la linea o le linee in questione).

Insert Space L'opzione Insert Space inserisce uno spazio in modo che possiate aggiungere un carattere nella posizione attuale del cursore.

Delete Char L'opzione Delete Char elimina il carattere che si trova nella posizione del cursore.

Insert Line L'opzione Insert Line aggiunge una linea vuota nella posizione attuale del cursore, in modo che possiate battere un'altra linea (di solito un'altra linea di programma).

Remove Line L'opzione Remove Line cancella l'intera linea su cui si trova il cursore.

Page Up L'opzione Page Up va alla pagina precedente del programma nella finestra Edit.

Page Down L'opzione Page Down va alla pagina successiva del programma nella finestra Edit.

Load Text L'opzione Load Text carica un testo dal drive e lo inserisce alla posizione del cursore.

Save Text L'opzione Save Text salva il testo specificato sotto il nome di file che specificherete.

New Buffer L'opzione New Buffer cancella l'attuale "contenitore" in memoria per il testo che avete appena caricato.

List L'opzione List dà il listato dell'intero programma (equivalente: LIST).

MENÙ DEBUG

Tron L'opzione Tron stampa su schermo i numeri di ciascuna linea che il programma esegue. Ciò vi aiuta a localizzare aree problematiche del programma mentre esso sta girando (equivalente: TRON).

Troff L'opzione Troff disinserisce l'opzione Tron (equivalente: TROFF).

Trace Mentre ciascuna linea del programma viene eseguita, l'opzione Trace stampa sullo schermo l'intera linea (equivalente: TRACE).

Untrace Disinserisce l'opzione Trace (equivalente: UNTRACE).

ALTRE ISTRUZIONI

Ci sono anche altre istruzioni nel BASIC ST che vi permettono di aprire, chiudere o cancellare ognuna delle quattro finestre. Queste istruzioni sono OPENW (per aprire una finestra), CLOSEW (per chiudere e far sparire una finestra) e CLEARW (per cancellare il contenuto di una finestra). I numeri da usare per indicare le finestre sono:

0 Edit
1 List
2 Output
3 Command

Se volete cancellare la finestra Output, basta battere *CLEARW* 2; chiuderete le finestre Edit, List e Command battendo:

```
CLOSEW 0: CLOSEW 1: CLOSEW 3
```

(Notate i due punti; essi separano i comandi.) Inoltre se volete che una certa finestra si dilati per tutto lo schermo (come quando si punta col mouse la casella d'ingrandimento della finestra), battete *FULLW* seguito dal numero della finestra.

Imparerete altre cose riguardo a queste operazioni quando esaminerò le singole parole chiave. Man mano che procedete nella lettura del libro, impegnatevi a provare i programmi e le linee di comandi campione; tale esperienza diretta vi faciliterà l'apprendimento e lo renderà molto più divertente.

PROGRAMMAZIONE ORIENTATA ALL'UTENTE

Questo è il momento giusto per parlare di *orientamento* di un programma verso l'utente. È *orientato all'utente* un programma che chiunque sarebbe in grado di usare; il programma non contiene nessun bug (cioè errori di programmazione) che provochino delle distorsioni nel suo funzionamento, le istruzioni nel programma e nella documentazione scritta sono chiare indipendentemente da quale errore, per quanto madornale, l'utente commetta, il computer perdona tutto e non esegue comandi disastrosi come cancellare un disco o riavviare il computer.

In altre parole, un programma orientato all'utente è "a tenuta stagna" contro l'errore dell'uomo e della macchina. Creare un programma orientato all'utente non è poi così difficile come sembra, ma dovrete essere meticolosi nel

provarlo. Inoltre, dovreste rendere il programma orientato anche all'utente programmatore, inserendo generosamente istruzioni REM (*remark*, ovvero commenti per voi stessi, che il computer ignorerà) nel corpo del programma, in modo che, anche se doveste esaminarlo ad anni di distanza, sareste ugualmente in grado di capire ciò che ogni sezione del programma svolge. Ciò rende il programma più facile da depurare dagli errori, sia nel breve che nel lungo termine.

A questo punto passiamo ad esaminare come si usano le più imortanti parole chiave del BASIC ST.

STAMPARE SULLO SCHERMO

Una della parole più facili del linguaggio, l'istruzione PRINT, stampa qualcosa sullo schermo (non sulla stampante; imparerete i comandi per quella nel Capitolo 7):

PRINT *espressione*

Qualsiasi elemento facente parte della funzione di una parola chiave, ma che non coincide con la parola chiave in sè, si definisce **argomento.** Nel nostro esempio, l'argomento espressione che specificate può essere formato da numeri, lettere e caratteri speciali. Dato che il cursore dovrebbe ancora trovarsi nella finestra Command, battete la linea seguente, poi premete una volta il tasto Return:

```
PRINT "HELLO, BABY"
```

Dopo aver premuto Return, le parole *HELLO, BABY* dovrebbero comparire sullo schermo nella finestra Output. Quello che avete fatto è molto semplice: PRINT ha dato istruzione al computer di stampare qualcosa sullo schermo, e le parole *HELLO, BABY* (racchiuse tra virgolette) erano ciò che volevate stampare.

Le virgolette indicano al computer che state per inserire una *stringa alfanumerica* (cioè una sequenza di caratteri, quali lettere, cifre o simboli) e non una *variabile numerica* (la rappresentazione di un numero il cui valore può cambiare nel corso del programma) e neppure una *variabile di stringa* (il nome assegnato ad una stringa il cui contenuto può variare).

Come vedrete, avreste anche potuto inserire un *valore numerico letterale*, che consiste semplicemente in un numero che non è una rappresentazione simbolica di qualcos'altro.

Vediamo alcuni esempi di come l'istruzione PRINT può stampare le stringhe:

```
PRINT "Questo e' un test"
```
Questo e' un test
```
PRINT "Potete separare le istruzioni":PRINT "con i due punti."
```
Potete separare le istruzioni
con i due punti.
```
PRINT "Potete separare le istruzioni ";:PRINT "con i due punti."
```
Potete separare le istruzioni con i due punti.

Nella seconda istruzione PRINT, i due punti (:) separavano le due diverse istruzioni PRINT; se non aveste battuto i due punti, il computer, non avrebbe accettato la linea ed avrebbe invece stampato questo:

Something is wrong
(Qualcosa non va)

Dovete sempre stare attenti a battere esattamente il formato che il computer richiede, poiché il computer segue alla lettera le vostre istruzioni e non potrà mai intuire ciò che realmete volete che faccia. La terza linea non solo mostra come i due punti possono separare due istruzioni diverse, ma anche come il punto e virgola (;) che segue un'istruzione PRINT (all'esterno delle virgolette, poiché quello che si trova all'interno di esse viene stampato tale e quale) dice al computer di non saltare alla linea successiva per l'istruzione PRINT che segue.

Stringhe

Prima di continuare con altri tipi d'istruzioni PRINT, è meglio che abbiate le idee chiare su cosa è una stringa. Una *stringa* è semplicemente una serie di caratteri raggruppati insieme. Tutte quelle che seguono sono stringhe:

Atari
1 2 3
Salve, sono una stringa
Ho venduto 3 robot oggi!!!

Non importa che usiate numeri, lettere o simboli, salvo, naturalmente, le virgolette.

Le stringhe, tuttavia, non sono solo voci tra virgolette che seguono l'istruzione PRINT. Possono anche essere registrate in memoria, rappresentate da variabili di stringa. Le *variabili di stringa* sono rappresentazioni di stringhe e sono seguite dal simbolo del dollaro ($), che sta ad indicare che sono ap-

punto variabili *di stringa*, e non variabili numeriche, le quali non riportano il simbolo del dollaro. Vediamo alcuni esempi di variabili di stringa:

```
A$
BB$
STRING$
HELLO$
A3$
P9$
```

Queste variabili di stringa non sono le stringhe vere e proprie: esse rappresentano le stringhe. Di conseguenza, la variabile A$ potrebbe rappresentare la stringa "Questo è un test". Si assegna questa stringa alla variabile battendo quello che segue:

```
A$="Questo e' un test"
```

Una volta che avrete premuto Return dopo aver inserito questa linea, la stringa "Questo è un test" verrà registrata in memoria e rappresentata dalla variabile A$. Di conseguenza, se battete:

```
PRINT A$
```

comparirà l'espressione *Questo è un test.* Avreste potuto ottenere lo stesso risultato battendo:

```
PRINT "Questo e' un test"
```

Vediamo un altro esempio:

```
PRINT A$;A$;A$
```
Questo e' un testQuesto e' un testQuesto e' un test

Il punto e virgola separava le variabili A$, ma avrete notato che non ci sono spazi tra le stringhe nell'output. Potreste inserire uno spazio tra una ripetizione e l'altra delle variabili inserendo una coppia di virgolette con uno spazio tra di loro per ciascuna ripetizione:

```
PRINT A$" ";A$" ";A$
```
Questo e' un test Questo e' un test Questo e' un test

Un'altra cosa utile da sapere è che una virgola inserita tra due variabili dice

al computer di andare verso destra dello spazio di un Tab (tabulazione). Provate:

```
PRINT A$,A$,A$
Questo e' un test          Questo e' un test          Questo e' un test
```

Le stringhe sono importanti e richiedono una trattazione accurata. Ci ritorneremo nel Capitolo 3.

Numeri

L'ultimo gruppo di elementi che si possono stampare con l'istruzione PRINT sono i numeri. Potete stampare direttamente un numero facendolo precedere da PRINT:

```
PRINT 300
300
```

Potete anche eseguire operazioni matematiche con PRINT ed il risultato comparirà sotto l'istruzione:

```
PRINT 3+5:PRINT 10*50:PRINT 600/12
8
500
50
```

I simboli matematici (+ per l'addizione, — per la sottrazione, * per la moltiplicazione e / per la divisione) operano tutti con i numeri e con PRINT, e anche le altre funzioni matematiche (che saranno trattate nel Capitolo 3) operano con l'istruzione PRINT.

Il punto interrogativo

Dato che batterete PRINT molto spesso quando inizierete a lavorare su programmi più grandi, potrete ricorrere all'abbreviazione di PRINT, che è il punto di domanda (?).

Esso avrà esattamente gli stessi effetti di PRINT quando lo usate nello stesso modo in cui usereste l'istruzione PRINT. Per esempio:

```
? "Ciao"
```

fa esattamente la stessa cosa di

```
PRINT "Ciao"
```

Regole per usare PRINT

Vediamo alcune regole da non dimenticare per usare l'istruzione PRINT:

1. I due punti (:) possono essere usati per separare qualsiasi tipo di istruzioni o comandi, comprese le istruzioni PRINT.

2. Quando fate seguire un punto e virgola (;) all'istruzione PRINT, il computer non salterà alla linea successiva per eseguire la successiva istruzione PRINT. Una virgola (,) dopo un'istruzione PRINT porterà l'espressione successiva da stampare verso destra di 13 spazi (cioè al Tab successivo).

3. Qualsiasi espressione inclusa tra le virgolette verrà stampata esattamente come si presenta. I segni speciali, come i due punti, il punto e virgola ed altri comandi non vanno collocati all'interno delle virgolette. Prestate anche attenzione a come usare le virgolette a coppie: le virgolette di apertura e le virgolette di chiusura. Il computer non saprà che cosa fare se incontra tre volte le virgolette in un'istruzione PRINT.

VARIABILI ED INPUT

Le variabili numeriche sono ancora più semplici da capire rispetto alle variabili di stringa. Una variabile numerica, come A, SS o K1, è uguale ad una variabile di stringa nel senso che rappresenta qualcos'altro. Ma, invece di rappresentare una stringa, rappresenta un numero. Per esempio, supponiamo che la variabile X sia uguale a 57.7. Se emetteste l'istruzione

```
PRINT X
```

comparirebbe il numero 57.7. Avere una lettera che rappresenta un numero ci ricorda l'algebra, ma la vera differenza è che il valore della variabile può cambiare.

Per assegnare un numero ad una variabile numerica, basta battere il nome della variabile, un segno di uguale (=) e poi il numero. Per esempio, se vogliamo che la variabile A sia uguale a 300, batteremo:

A = 300

Quasi tutte le versioni di BASIC, compreso il BASIC ST, usano la parola chiave opzionale LET davanti a questo genere di definizione di variabile. Invece di battere semplicemente **A = 300**, potrete battete **LET A = 300.** Se usare LET facilita la comprensione del programma, usatela. Tuttavia la maggior parte dei programmatori omettono l'istruzione LET.

Potete dare un nome qualsiasi ad una variabile numerica (lo stesso vale anche per una variabile di stringa). Vediamo solo alcune regole da seguire:

- Assicuratevi di non usare una parola chiave come nome di una variabile o all'interno di esso. La parola *PRINT*, per esempio, non può costituire un nome di variabile, e nemmeno XXXPRINTXXXX, poiché questo nome comprende la parola.
- Il primo carattere del nome di una variabile deve essere una lettera, e non un numero; potete tuttavia impiegare dei numeri all'interno del nome di una variabile. Alcuni nomi possibili sono AAA, G99, XX4, L e R2R. Non usate niente oltre a lettere e numeri, per esempio evitate &, * o #.
- È spesso utile assegnare ad una variabile un nome che ne rifletta la funzione. Per esempio, se battete il programma di un gioco e dovete memorizzare il punteggio del giocatore 1 in una variabile, potreste chiamare la variabile PUNTEGGIO1 o GIOCATORE1. Poiché potete usare fino a 31 caratteri nei nomi di variabile, c'è abbastanza spazio perché il nome che le date sia allusivo alla sua funzione.

Le variabili numeriche, come le variabili di stringa, sono flessibili. Potete porre una variabile uguale ad un'altra variabile, uguale ad una equazione matematica, o uguale a se stessa addizionata ad un altro numero o variabile. Tratterò le variabili, sia numeriche che di stringa, con maggiori dettagli nel Capitolo 3.

INPUT e LINE INPUT

Altre due istruzioni con cui dovete fare la conoscenza sono INPUT e LINE INPUT. Queste due istruzioni permettono al computer di ricevere l'input dalla tastiera. INPUT, quando è seguito dal nome di una variabile, fermerà il programma e porrà un punto di domanda sullo schermo, in attesa che inseriate qualcosa. Se INPUT è seguito da una variabile numerica, dovrete inserire un numero; se INPUT è seguito da una variabile di stringa (cioè che riporta una segno di dollaro), il computer accetterà una stringa dalla ta-

stiera. Il loro formato sintattico è quello che segue:

INPUT*[''prompt'';]variabile*[,variabile, ...]
LINE INPUT*[''prompt'';]variabile di stringa*

Vediamo alcuni esempi:

INPUT X
? 15.5
PRINT X
15.5
INPUT AA$
? Ehi la'
PRINT AA$,X
Ehi la'   15.5

Come al solito dovrete premere il tasto Return per dire al computer che avete completato la linea da inserire. Per chiarire lo scopo del punto di domanda (e per rendere il programma più interattivo e quindi orientato all'utente, potreste inserire una frase tra la parola chiave INPUT ed il nome della variabile; questo permette al computer di ''porvi una domanda'' in modo da sapere cosa si deve inserire. Vediamo alcuni esempi d'input di variabili numeriche e di stringa, ciascuno dei quali presenta un *prompt:*

INPUT ''Inserisci la tua eta''';ETA
Inserisci la tua eta'? 20
INPUT''Come ti chiami'';NME$
Come ti chiami? Tim
PRINT ETA,NME$
20   Tim

La versatilità di INPUT vi permette anche di ottenere più di un dato per volta. In effetti si possono mescolare variabili numeriche e di stringa in una sola linea INPUT. Per esempio:

INPUT''Inserisci il nome del prodotto, il codice e la quantita' ordinata'', NME$,C$,Q
Inserisci il nome del prodotto, il codice e la quantita' ordinata
cilindretti,22X3,7000
PRINT C$,Q,NME$
22X3   7000   cilindretti

Questo esempio illustra diversi punti importanti dell'istruzione INPUT e del BASIC ST. Notate innanzitutto che dovete inserire i dati nell'ordine in cui il computer se li aspetta: l'istruzione INPUT chiede una stringa, poi un'altra stringa, quindi un numero ed i dati dalla tastiera devono essere inseriti nello stesso ordine. Potete anche notare, tuttavia, che una volta che i valori sono stati assegnati alle variabili, il computer può operare su di essi in un ordine qualsiasi; in effetti non è necessario che lavori con tutte contemporaneamente. Notate inoltre che si devono impiegare le virgole sia nell'istruzione INPUT, sia nell'input dell'utente per separare i valori delle variabili fornite. Infine c'è da notare che il prompt non contiene il punto di domanda. Dato che alcuni prompt INPUT non sono domande, il comando può essere inserito con una virgola anziché col punto e virgola, così facendo si sopprime il punto interrogativo.

LINE INPUT è una versione più specializzata di INPUT, dato che accetta solo una variabile per volta; tuttavia, accetta virgole, punti interrogativi ed altri segni di punteggiatura che altererebbero invece il funzionamento di INPUT. È opportuno ricorrere a LINE INPUT quando pensate che la stringa da inserire possa contenere alcuni dei suddetti segni di punteggiatura; qualsiasi cosa vi si inserisca, verrà collocata nella variabile di stringa, compresi punti interrogativi, virgole e così via.

NUMERI DI LINEA

Un qualsiasi programma scritto in un linguaggio qualsiasi si compone di un gran numero di linee che il computer segue in un determinato ordine. Mentre potete battere le istruzioni direttamente nella finestra Command e vederne i risultati nella finestra Output, non potrete invece vedere l'esito delle linee di un programma finché non lancerete l'intero programma.

Un modo di stabilire l'ordine tra le molte linee di un programma è di dare a ciascuna un numero di linea. Mentre non è necessario che battiate le linee nel loro ordine definitivo, è invece necessario che diate al computer dei numeri di linee appropriati, in modo che sappia come le dovrà ordinare (è l'ST che le riordina man mano che le battete). Immaginiamo che battiate le linee seguenti nella finestra Command (potete farlo anche subito; uno dei modi più efficaci d'imparare con questo libro è di battere, usare e modificare i programmi campione):

```
30 PRINT "E' tutto per ora!"
10 PRINT "Questo e' un programma semplice."
20 PRINT "Infatti usa solo un'istruzione."
```

Ogni istruzione è preceduta da un numero di linea (anche se potete combinare più istruzioni su una sola linea, separandole con i due punti) che può essere qualsiasi numero intero positivo da 1 a 65535. Molte persone normalmente numerano le linee 10, 20, 30, 40, 50 e così via, perché in tal modo rimane lo spazio per inserire linee in un secondo tempo, tra quelle già esistenti. Per vedere questo programma, mandatelo alla finestra List battendo *LIST* e poi (come al solito) premendo il tasto Return:

```
LIST
10 PRINT "Questo e' un programma semplice."
20 PRINT "Infatti usa solo un'istruzione."
30 PRINT "E' tutto per ora!"
```

Potete notare che le linee adesso sono ordinate. Questo è un programma in BASIC. Potete vedere quello che fa (che probabilmente non costituirà una gran sorpresa) battendo il comando *RUN* e premendo il tasto Return. Le tre linee di testo compariranno nella finestra Output quasi immediatamente.

AUTO

Quando state scrivendo un programma lungo, vi stancherete presto di battere i numeri di linea, specialmente se seguono una successione regolare (5, 10, 15, 20 ... o qualcosa di simile).

Il comando AUTO affida al computer la numerazione delle linee mentre voi dovrete batterne il contenuto. AUTO è utile solo se state battendo una parte di programma nello stesso ordine che poi le linee dovranno seguire. Il formato di questo comando è:

AUTO *inizio,incremento*

dove *inizio* è il numero di linea da cui cominciare e *incremento* dice di quanto aumenta ciascun numero rispetto al precedente. AUTO 10,20, per esempio, produrrà i numeri di linea 10, 30, 50, 70 e così via, finché non vorrete uscire dalla numerazione automatica premendo Control-G. Il computer aspetterà che inseriate il contenuto della linea e che premiate Return prima di stampare sullo schermo il numero di linea successivo. Infine, se il computer visualizza un numero di linea che già esiste, compariranno davanti al numero due asterischi (**); se volete battere una linea nuova che cancella la precedente, basta procedere normalmente. Se invece non volete cancellare la vecchia linea, interrompete il comando AUTO con Control-G.

Riferimenti di linea: GOTO ed Etichette

I numeri di linea non si limitano ad offrire un modo per ordinare le diverse linee di comando, ma servono anche come punti di riferimento in base ai quali il ST possa saltare in diversi punti del programma. Vediamo un breve esempio che illustra questo concetto:

```
10 PRINT "Spero che la prossima frase ti piacera':"
20 PRINT "Di chi e' questo caffe'?"
30 GOTO 20
```

Quando lancerete questo programma, l'output sarà:

Spero che la prossima frase ti piacera':
Di chi e' questo caffe'?
Di chi e' questo caffe'?
Di chi e' questo caffe'?
(e così via)

Questo si definisce loop infinito, perché il computer non ne uscirà mai. GOTO 20 della linea 30 dice al computer di ritornare all'inizio della linea 20 e di ricominciare da lì.

Poiché continuerà ad eseguire la linea 30 dopo aver eseguito la linea 20, non smetterà mai di ripetere la sua innocua domanda. Come vedete, i numeri di linea sono usati come riferimento per dare al computer dei punti precisi dove andare. L'istruzione GOTO ha due possibili formati sintattici:

GOTO *numero di linea*
GOTO *etichetta*

Come potete vedere, un altro possibile riferimento è l'*etichetta.* L'etichetta è semplicemente il nome che date ad una linea del programma, esso, seguito dai due punti (:), vi offre un sistema ancora più facile di far riferimento ad una certa routine. Quando scrivete un programma lungo, non è pratico dover ricordare un lungo elenco di numeri di linea e delle routine corrispondenti.

Invece potrete dare alle routine delle specifiche etichette che possiate facilmente ricordare. Il seguente programma campione accetta dalla tastiera informazioni sul tipo di computer che avete (se la risposta è Atari ST, il programma si congratulerà con voi; in caso contrario aspettatevi pure una critica):

```
10 INIZIO:PRINT "Questo e' il testo del computer"
20 INPUT "Che tipo di computer possiedi";COMP$
30 IF COMP$ = "Atari ST" THEN GOTO BENE
40 PRINT "E' proprio vero che i gusti sono gusti.":GOTO INIZIO
50 BENE: PRINT "Buon per te. Ottima scelta, se cosi' posso
   esprimermi."
60 GOTO INIZIO
```

L'etichetta BENE serve da punto di riferimento per la routine che si congratula con l'utente e INIZIO segna l'inizio del programma. Se lanciate il programma, l'output potrebbe essere:

Che tipo di computer possiedi? ZIPCHIP 80 Model 1
È proprio vero che i gusti sono gusti.
Che tipo di computer possiedi? Atari ST
Buon per te. Ottima scelta, se cosi' posso esprimermi.

GOSUB

Talvolta serve che il computer vada ad una certa routine e che da essa ritorni una volta che ha finito di usarla. Ciò capita molto spesso quando si hanno routine predefinite che ricorrono nel programma in diversi punti (una simile routine si definisce anche *subroutine*). L'istruzione GOSUB funziona come GOTO, nel senso che manda il computer ad una certa linea o etichetta del programma (che potremo chiamare col suo termine inglese 'label'), ma quando incontra la parola chiave RETURN, il computer ritorna al punto in cui si trovava prima di andare alla subroutine. La sintassi di questa struttura è:

GOSUB *numero di linea o etichetta*

.

.

.

RETURN

L'esempio che segue e l'ouput che ne risulta serviranno a chiarire il concetto:

```
10 PRINT "Dobbiamo imparare altre istruzioni BASIC."
20 GOSUB 30:GOTO 10
30 PRINT "Certamente, per esempio GOSUB e RETURN!"
40 RETURN
RUN
```

Dobbiamo imparare altre istruzioni BASIC.
Certamente, per esempio GOSUB e RETURN!
Dobbiamo imparare altre istruzioni BASIC.
Certamente, per esempio GOSUB e RETURN!
Dobbiamo imparare altre istruzioni BASIC.
Certamente, per esempio GOSUB e RETURN!

Quando il computer incontra GOSUB 30 nella linea 20, salta alla linea 30 ricordando l'istruzione successiva. Quando incontra l'istruzione RETURN nella linea 40, il computer ritorna a quel punto. Ogni GOSUB deve incontrare un RETURN, altrimenti non può ritornare nel punto che aveva lasciato.

Potete anche modificare uno dei programmi precedenti in modo che continui a svolgere la stessa funzione di prima, ma questa volta usando GOSUB e RETURN, invece di GOTO:

```
10 INIZIO:PRINT "Questo e' il testo del computer"
20 INPUT "Che tipo di computer possiedi";COMP$
30 IF COMP$="Atari ST" THEN GOSUB BENE:GOTO INIZIO
40 PRINT "E' proprio vero che i gusti sono gusti.":GOTO INIZIO
50 BENE: PRINT "Buon per te. Ottima scelta, se cosi' posso
   esprimermi."
60 RETURN
```

ON

Una variazione alle istruzioni GOTO e GOSUB ci viene offerta dalla parola chiave ON, che rende GOTO e GOSUB molto più versatili. Il formato per usare ON è:

ON *variabile* GOTO *linea o etichetta[,linea o etichetta,...]*
ON *variabile* GOSUB *linea o etichetta[,linea o etichetta,...]*

Il comando ON dice al computer "Se la variabile è uguale a 1, vai al primo numero di linea; se la variabile è uguale a 2, vai al secondo numero di linea; se la variabile è uguale a 3, vai al terzo numero di linea; e così via. Potete dirigere i corrispondenti numeri di linea su GOTO o GOSUB e potete elencare tutti numeri di linea o tutte le etichette che volete dopo le parole chiave GOTO e GOSUB. Vediamo un breve programma che illustra in che modo questo comando può rivelarsi utile. Prima vediamo un programma inefficiente:

```
 5 INPUT "Dammi un valore tra 1 e 4 per la variabile";A
 7 IF A<1 OR A>4 THEN GOTO 5
10 IF A=1 THEN GOTO 50
20 IF A=2 THEN GOTO 60
30 IF A=3 THEN GOTO 70
40 IF A=4 THEN GOTO 80
50 PRINT "A E' UNO":END
60 PRINT "A E' DUE":END
70 PRINT "A E' TRE":END
80 PRINT "A E' QUATTRO":END
```

Ora vediamo lo stesso programma reso più efficente con ON:

```
 5 INPUT "Dammi un valore tra 1 e 4 per la variabile";A
 7 IF A<1 OR A>4 THEN GOTO 5
10 ON A GOTO 50,60,70,80
50 PRINT "A E' UNO":END
60 PRINT "A E' DUE":END
70 PRINT "A E' TRE":END
80 PRINT "A E' QUATTRO":END
RUN
Dammi un valore tra 1 e 4 per la variabile? 3
A E' TRE
```

Noterete quanto più corto sia il secondo programma rispetto al primo. Quando elencate i numeri di linea dopo ON...GOTO o ON...GOSUB, non è necessario che siano in ordine. Potreste scrivere:

```
ON T GOSUB 50, 23, 600, 4, 700
```

Infine, potete usare l'istruzione ON ERROR GOTO seguita da un numero di linea o da un'etichetta per dire al computer di andare a un determinato punto del programma se incontra un errore.

I LOOP

Un altro concetto fondamentale che dovete capire per programmare in BASIC è quello di *loop* e del suo funzionamento. Un loop non è altro che una serie d'istruzioni che verranno eseguite ripetutamente in un programma. Il numero delle ripetizioni può essere determinato semplicemente contando da un numero al successivo, oppure provando una certa condizione finché questa

non si verifichi. Prima esaminiamo il primo caso. Capita spesso che un programma debba contare, per esempio per generare una pausa nel programma o per tracciare un grafico. Quando un computer (o una persona) conta, inizia da un numero e finisce con un altro. Per andare da uno all'altro, conta con un dato incremento. Per esempio, dovendo scrivere un programma che conti da 1 a 11 con incremento 2, stampando man mano ciascun numero, potreste scrivere:

```
10 A=1
20 PRINT A
30 A=A+2
40 GOTO 20
```

FOR...NEXT

Un modo più efficente di fare la stessa cosa, tuttavia, è ricorrere al loop FOR...NEXT:

FOR *indice=inizio* TO *fine* [STEP *incremento*]
    corpo loop
NEXT

Questo dice al computer di contare da un certo numero (*inizio*) ad un altro numero (*fine*) ripetendo le istruzioni contenute nel *corpo del loop* con l'*incremento* specificato. Nell'esempio precedente, potreste riscrivere il programma in una sola linea:

```
10 FOR A=1 TO 11 STEP 2:PRINT A:NEXT
```

Usando la variabile *indice* A, il computer sa che deve cominciare a contare da 1, smettere di contare a 11 e contare con un incremento di 2. Omettendo la clausola opzionale STEP, conterà con incrementi di 1.

Il loop FOR...NEXT è versatile, poiché potete sostituire i numeri con le variabili, usare numeri negativi e finire il loop con l'istruzione NEXT quando volete. Quello che segue è un esempio più complesso del loop FOR...NEXT:

```
10 A=500:YC1=45.5
20 FOR X=A TO YC1 STEP —.5
30 PRINT "La variabile X e' attualmente ";X
40 NEXT
RUN
```

La variabile X e' attualmente 500
La variabile X e' attualmente 499.5
(e così via fino a)
La variabile X e' attualmente 45.5

Così come i numeri di linea possono svolgere due funzioni, ordinare le linee e servire come punti di riferimento, i loop FOR...NEXT svolgono anch'essi due funzioni: non solo contano, il che vi permette di ripetere una certa funzione un qualsiasi numero di volte, ma le variabili indice che impiegano per contare possono essere usate nel programma. Mentre il loop FOR...NEXT conta, potete usare per qualche altro scopo le variabile con cui sta contando. Immaginiamo che abbiate scritto un programma che accetti dalla tastiera un numero iniziale e uno finale e poi stampi i quadrati di tutti i numeri compresi tra i due. Il programma potrebbe essere qualcosa di questo genere:

```
10 INPUT FRST:INPUT LST
20 FOR X=FRST TO LST STEP 1
30 PRINT "Il quadrato di";X;"e'";X*X
40 NEXT
RUN
? 4
? 6
Il quadrato di 4 e' 16
Il quadrato di 5 e' 25
Il quadrato di 6 e' 36
```

Nidificazione

Un'altra tecnica, chiamata *nidificazione*, serve quando avete un loop all'interno di un altro loop. Ricordate che il loop FOR...NEXT non è che un dispositivo di numerazione e possiamo avere un contatore all'interno di un altro contatore (nonché un contatore all'interno di un altro contatore ancora all'interno di un contatore e così via). Diamo un'occhiata a questo programma e all'output che produce:

```
10 FOR A=1 TO 3
20 FOR B=1 TO 4
30 PRINT A;",";B,
40 NEXT:PRINT:NEXT
RUN
```

1,1 1,2 1,3 1,4
2,1 2,2 2,3 2,4
3,1 3,2 3,3 3,4

Il loop FOR...NEXT che impiega la variabile A conta da 1 a 3, ma ogni volta che questo loop viene incrementato, il loop che impiega la variabile B conta da 1 a 4. Il loop B è annidato nel loop A, quindi ogni volta che gira il loop A, il loop B svolge la sua intera sequenza di numerazione.

Vediamo un altro esempio:

```
10 FOR A=1 TO 3
20 PRINT "Io sono il loop A"
30 FOR B=1 TO 2:PRINT "Io sono il loop B":NEXT
40 NEXT
RUN
```

Io sono il loop A
Io sono il loop B
Io sono il loop B
Io sono il loop A
Io sono il loop B
Io sono il loop B
Io sono il loop A
Io sono il loop B
Io sono il loop B

Una volta che il loop finisce, procede con qualsiasi cosa segua. In questo caso, il programma è finito, quindi il controllo ritorna un'altra volta alla finestra Command. Per rendere il listato del programma meno confuso, potete completare l'istruzione NEXT con la corrispondente variabile che il loop impiega. Con NEXT A e NEXT B, potete stabilire con facilità quale dei due corrisponde a ciascun FOR.

WHILE...WEND

Un altro tipo di loop, che impiega le parole chiave WHILE e WEND si differenzia dal loop FOR...NEXT perché si tratta di un loop *condizionale*, vale a dire che invece di contare da un numero ad un altro, il loop continua a girare finché non si è verificata una certa condizione. Quando tale condizione si verifica, il computer procede a quello che segue (in altri termini, qualsiasi cosa venga dopo WEND che designa la fine del loop). Il formato è simile a quello di FOR...NEXT:

```
WHILE espressione
  corpo del loop
WEND
```

Per impostare un loop WHILE...WEND, aggiungete alla parola WHILE la condizione (*espressione*) che si deve verificare perché il loop sia completo. Quindi inserite qualsiasi azione che volete venga svolta (*corpo del loop*) finché la condizione continua a *non* essere vera (in altre parole finché il loop gira). Chiudete il loop con un'istruzione WEND. Nell'esempio che segue, il loop racchiude un'istruzione INPUT che chiede il nome dell'utente; salvo che l'input corrisponda a Francesco Cossiga, il loop continuerà a girare: quando tuttavia la suddetta condizione si verifichi, il computer stampa sullo schermo *Buon giorno, Signor Presidente.*

```
10 FULLW 2:CLEARW 2
20 WHILE NME$<>"Francesco Cossiga"
30 INPUT "Come ti chiami";NME$
40 WEND
50 PRINT "Buon giorno, Signor Presidente."
```

Il loop WHILE...WEND può anche essere annidato, dandovi la possibilità di stabilire loop condizionali più complessi. Cercate di usare i loop FOR...NEXT e WHILE...WEND nel modo più efficiente possibile; ne troverete abbondanti esempi in tutto il libro.

MATRICI

Anche se le variabili numeriche e di stringa sono un modo pratico di immagazzinare informazioni, può capitare di avere dati che richiedano un sistema di definizione più organizzato. Per esempio, supponiamo che abbiate i dati relativi a tre agenti di vendita, per un periodo di tre mesi, e vogliate registrare l'ammontare in dollari delle vendite per ciascun mese. Chiamiamo questi ipotetici agenti Bruno, Alfredo e Carlo e registriamo le loro vendite per i mesi di Febbraio, Marzo e Aprile. Volendo registrare questi dati numerici con variabili ordinarie, potremmo presentarle come segue:

```
BF=500      AF=450      CF=690
BM=525      AM=625      CM=333
BA=452      AA=533      CA=546
```

Questa serie di variabili apparentemente prive di qualsiasi relazione tra loro è evidentemente un modo inefficiente di memorizzare i dati, specialmente quando cominciate ad avere 20 venditori per un periodo di 30 mesi. Con un gruppo piccolo come il nostro le variabili sono relativamente facili da ricordare (AM sta per Alfredo in Marzo, CA sta per Carlo in Aprile e così via), ma fare calcoli con queste variabili diventerà presto pesante.

Il modo più efficiente di memorizzare questo genere d'informazioni ordinate è ricorrere ad una matrice.

Una *matrice*, che in altri linguaggi si chiama tabella, è un insieme di variabili numeriche e di stringa disposte in modo ordinato.

Poiché nel nostro caso stiamo trattando due ordini d'informazioni (agenti di vendita e mesi), assegnamo dei numeri a ciascuna di queste *dimensioni*. Gli agenti avranno i seguenti numeri: Bruno=1, Alfredo=2 e Carlo=3 ed i mesi saranno Febbraio=1, Marzo=2 e Aprile=3. Indicando la matrice con la variabile V, le vendite di Bruno in Marzo si potranno indicare con V(1,2), dato che il numero corrispondente a Bruno è 1 e il numero per Marzo è 2. Le vendite di Carlo in Febbraio verranno registrate sotto la variabile V(3,1). Avremo in tutto nove variabili da V(1,1) a V(3,3) e con questo sistema sarà molto facile reperire l'ammontare delle vendite di ciascuno per un certo mese.

Si dice che le matrici hanno un certo numero di dimensioni. Nel nostro caso erano due, gli agenti di vendita e i mesi, quindi la nostra è una matrice bidimensionale. Una matrice ad una dimensione può essere esemplificata dai voti di uno studente sui 12 mesi, per esempio da G(1) a G(12). Una matrice a due dimensioni è dello stesso tipo dell'esempio che abbiamo visto prima, ed impiega variabili del tipo V(3,2). Una matrice tridimensionale ha tre elementi, per esempio la registrazione delle vendite di un certo prodotto (dimensione uno) in un certo mese (dimensione due) in un dato negozio (dimensione tre); in tal modo registreremo le vendite del prodotto 2 nel mese 5 nel negozio 3 con la variabile V(2,5,3).

Il numero delle dimensioni trova un limite nella memoria del computer e quando si superano le tre dimensioni, la quantità di memoria richiesta diventa piuttosto grande. Tuttavia, con i 512K o i 1024K di memoria degli ST, soltanto una matrice molto grande provocherebbe un errore per insufficienza di memoria.

DIM

Per destinare una certa quantità di memoria ad una matrice, dobbiamo ricorrere all'istruzione DIM:

DIM *variabile(indice[,indice,...])*

DIM è facile da usare, in quanto dovete solo aggiungervi il numero massimo di dimensioni che volete stabilire per la vostra matrice. Per esempio, se volete impostare una matrice a tre dimensioni che vada da XX(0,0,0) a XX(5,5,3), basterà battere DIM XX(5,5,3) prima di assegnare qualsiasi valore agli elementi della matrice.

Userete DIM quasi sempre all'inizio di un programma e potrete impostare le dimensioni di una certa matrice di variabili solo una volta (non è possibile fare altrimenti).

OPTION BASE

Se preferite che la vostra matrice cominci da 1 anziché da 0, potete ricorrere all'istruzione OPTION BASE:

OPTION BASE *numero*

Se non usate OPTION BASE, qualsiasi matrice di variabili impostiate inizierà dall'elemento 0; quindi, se battete DIM A(5), avrete a disposizione A(0), A(1), A(2), A(3), A(4) e A(5). Molte persone tuttavia preferiscono iniziare dal numero 1 anziché da 0 e basterà battere OPTION BASE 1 perché la matrice inizi dall'elemento 1. OPTION BASE 1 seguito da DIM A(5) vi metterà a disposizione le variabili A(1), A(2), A(3), A(4) e A(5). Se volete tornare all'impostazione di default, basta battere OPTION BASE 0.

Matrici di stringa

Non dimenticate che le matrici non sono destinate solo ai numeri; per quanto i dati numerici si prestino meglio alla collocazione in matrice, si possono memorizzare stringhe in matrici di stringa. Alcuni esempi di nomi di matrici di stringa possono essere A$(3,3), BH$(5) e SCU1$(6,3,9). Non cambia niente nelle matrici di variabili di stringa rispetto alla matrici di variabili numeriche, tranne l'aggiunta del simbolo del dollaro.

Per esempio, potremo avere la seguente matrice di stringa unidimensionale:

```
A$(1)="Lombardia"
A$(2)="Toscana"
A$(3)="Sicilia"
PRINT A$(3)
Sicilia
```

COME LEGGERE I DATI

Se già avete un insieme di numeri o di stringhe che volete memorizzare in variabili, potete ricorrere alle istruzioni READ e DATA per inserire le informazioni senza doverle battere tutte le volte che lanciate il programma:

READ *variabile[,variabile]*

.

.

.

DATA *numero o stringa[,numero o stringa,...]*

READ e DATA sono facili da comprendere: READ leggerà una certa quantità di dati da un'altra parte del programma e voi designerete la localizzazione dei dati contrassegnandoli con l'istruzione DATA.

Potete usare il comando READ sia con variabili numeriche che di stringa e, purché vi assicuriate che le *variabili* nell'istruzione READ corrispondano con il tipo di ciascuna voce di DATA (*numero o stringa*) da leggere, non dovreste avere alcuna difficoltà. Per esempio, quanto segue darà luogo ad un errore, poiché nell'istruzione DATA le variabili numerica e di stringa sono invertite:

```
10 READ A,A$
20 DATA Hello,21
```

Ora invece vediamo un esempio corretto di lettura delle variabili A, B$ e CC:

```
10 READ A,B$,CC
20 PRINT "Lettura completa dei dati"
30 DATA 40,Hello,30
```

Quando poi il programma finisce, A sarà uguale a 40, B$ corrisponderà alla stringa Hello e CC sarà uguale a 30. Il prossimo esempio è più complesso. Innanzitutto la finestra Output viene ingrandita a tutto lo schermo ed il suo contenuto cancellato (vedi la linea 10 del Listato 2.1).

Successivamente, la struttura del loop annidato delle linee 20-50 legge i nove dati delle linee 130-150 del programma nella matrice SA(S,N). Dopo aver letto i dati, il ST legge i tre mesi nella matrice MN$(x).

I nomi, i mesi ed i numeri di codice vengono stampati in modo che sappiate quali numeri inserire per conoscere le vendite di ciascuna persona in un certo mese (vedi linee 60-100 del Listato 2.1).

```
10 FULLW 2: CLEARW 2
20 FOR S=1 TO 3
30 FOR N=1 TO 3
40 READ SA(S,N)
50 NEXT:READ NM$(S):NEXT:READ MN$(1),MN$(2),MN$(3)
60 PRINT "Agenti di vendita:"
70 PRINT "Bruno=1, Alfredo=2, Carlo=3"
80 PRINT "Mesi:"
90 PRINT "Maggio=1, Giugno=2, Luglio=3"
100 INPUT "Quali dati vuoi vedere",n1,n2
110 PRINT "Vendite di ";NM$(N1);" in ";MN$(N2);": Lit.";SA(N1,N2)
120 GOTO 60
130 DATA 540,430,612,Bruno
140 DATA 367,721,443,Alfredo
150 DATA 469,382,678,Carlo
160 DATA "Maggio","Giugno","Luglio"
```

Listato 2.1: uso di READ...DATA con le matrici.

Una volta che il computer ha entrambi i numeri che individuano l'ammontare di vendite da visualizzare, stampa l'informazione e ritorna alla linea 60 (vedi le linee 110-120). Le ultime linee del programma (130-160) sono istruzioni DATA e anche se potrebbero essere poste in un punto qualsiasi del programma, si trovano di solito nella parte finale.

Quando lanciate il programma del Listato 2.1, potete inserire il nome e il mese che vi interessano (usando i rispettivi numeri di codice); una volta che il computer riceve queste informazioni, può recuperare i dati da voi richiesti tra quelli che conserva in memoria:

```
RUN
Agenti di vendita:
Bruno=1, Alfredo=2, Carlo=3
Mesi:
Maggio=1, Giugno=2, Luglio=3
Quali dati vuoi vedere? 2,1
Vendite di Alfredo in Maggio: Lit. 367
```

Usando l'istruzione DATA potete impostare gruppi fissi d'informazioni e con l'aiuto delle matrici di variabili, i dati sono ancora più facili da compilare e presentare.

RESTORE

In alcuni programmi è necessario leggere gli stessi dati più di una volta. A questo scopo il ST prevede l'istruzione RESTORE:

RESTORE *[numero di linea]*

Ogni volta che il computer legge dei dati, nella sua memoria viene fissato un puntatore che indica i dati immediatamente successivi (altrimenti l'istruzione READ continuerebbe a leggere lo stesso primo numero). Il puntatore continua a spostarsi al dato successivo ogni volta che usa READ ed una volta che tutti i dati sono stati letti, riceverete un messaggio d'errore se cercate di continuare a leggere ancora. RESTORE riporta il puntatore al primo elemento dell'appropriata istruzione DATA, in modo che i dati possano essere letti un'altra volta. Il parametro opzionale *numero di linea* vi permette di specificare quale istruzione DATA rileggere; senza di esso, l'istruzione RESTORE riporta il puntatore alla prima istruzione DATA del programma. Nel Capitolo 4 userò RESTORE in questo modo per tracciare delle ellissi.

DECISIONI

La capacità del ST di prendere decisioni si rivelerà vitale nella stesura dei vostri programmi. Il computer deve essere in grado di operare delle scelte tra diverse possibili soluzioni basate su risultati certi. Il ST può decidere se un'affermazione è vera o falsa, se un numero è maggiore, minore o uguale ad un altro, se una stringa corrisponde ad un'altra e così via; in base al risultato di una decisione, il ST intraprende le azioni per le quali lo avete istruito.

IF..THEN

La parola chiave decisionale fondamentale nel BASIC ST è IF, che non fa altro che verificare se un'espressione è vera o falsa. Con diversi operatori logici, l'istruzione IF diventa abbastanza flessibile da prendere molte decisioni diverse. Il suo formato è:

IF *condizione* THEN *azione1* [ELSE *azione2*]

In linguaggio comune questa istruzione significa: "Se si verifica la prima

condizione, metti in atto la prima *azione*, altrimenti metti in atto la *seconda*".
Vediamo ora un programma che fa dell'ST un maldestro indovino dell'età:

```
10 CLEARW 2:FULLW 2
20 FOR AGE=1 TO 100
30 PRINT "La tua eta' e' ";AGE;"? Per favore rispondi SI o NO."
40 INPUT ANSWER$
50 IF ANSWER$="SI" OR ANSWER$="Si" OR ANSWER$="si"
   THEN GOTO 70
60 NEXT
70 PRINT "LA TUA ETA' E' ";AGE
RUN
La tua eta' e' 1? Per favore rispondi SI o NO.
NO
La tua eta' e' 2? Per favore rispondi SI o NO.
NO
La tua eta' e' 3? Per favore rispondi SI o NO.
SI
LA TUA ETA' E' 3
```

La linea 40 accetta l'input dalla tastiera. Se ANSWER$ è sì, il computer va alla linea 70, in base alla quale ripete l'età dell'utente. Se la condizione non

=	Il primo operatore è uguale al secondo operatore?
>	Il primo operatore è maggiore del secondo operatore?
<	Il primo operatore è minore del secondo operatore?
< >	Il primo operatore è diverso dal secondo operatore?
= >	Il primo operatore è maggiore o uguale al secondo operatore?
< =	Il primo operatore è minore o uguale al secondo operatore?
AND	La prima e la seconda condizione sono entrambe verificate?
OR	È verificata la prima o la seconda condizione?
EQV	La prima e la seconda condizione sono o ENTRAMBE vere o ENTRAMBE false?
NOT	Inverte qualsiasi test logico funzionale.

Tabella 2.1: operatori logici usati dopo IF.

è vera, il computer va alla linea successiva, che lo istruisce a continuare il loop con l'istruzione NEXT. Notate che la linea 50 potrebbe anche essere:

```
IF ANSWER$="SI" THEN GOTO 70 ELSE NEXT
```

Avevo tralasciato ELSE per dimostrare che il computer salta alla linea successiva se una condizione non si verifica, anche se non c'è un'istruzione ELSE a dargli un'alternativa. La *condizione* da verificare dopo l'istruzione IF è un semplice test logico. Potete confrontare stringhe, numeri e matrici di variabili, controllare un certo lasso di tempo, o qualsiasi altra cosa che possa essere verificata come vera o falsa entro i limiti del linguaggio del computer. Gli operatori logici che potete impiegare dopo IF sono elencati nella Tabella 2.1.

Il prossimo esempio di programma mostra come funzionano la maggior parte di queste funzioni. Ricordate che il programma è deliberatamente grezzo per metterne meglio in evidenza le funzioni; la maggior parte di questi programmi potrebbero essere riscritti più elegantemente in due o tre linee al massimo.

```
10 CLEARW 2:FULLW 2
20 INPUT "Inserisci il tuo primo numero";FN
30 INPUT "Inserisci il tuo secondo numero";SN
40 INPUT "Qual e' la tua eta' ";AGE
50 IF FN>SN THEN PRINT "Il tuo primo numero e' maggiore
   del secondo" ELSE IF FN<SN THEN PRINT "Il tuo primo numero
   e' minore del secondo" ELSE PRINT "Il tuo primo numero e'
   uguale al secondo"
60 IF FN=AGE AND SN=AGE PRINT "Il tuo primo numero,
   il secondo numero e la tua eta' sono identici.":GOTO 80
70 IF FN=AGE OR SN=AGE PRINT "Il primo numero o il secondo
   sono uguali alla tua eta'."
80 PRINT "Questo e' tutto per ora. Sono stanco di prendere decisioni."
```

L'attività decisionale è importante in tutta la programmazione, grafici ed effetti acustici compresi. Un programma utile si basa quasi sempre sulla sua capacità di agire sotto certe condizioni e la struttura IF..THEN...ELSE è quanto vi serve per passare ad altre parti diverse del programma una volta che una decisione sia stata presa.

EDITING

Anche i migliori programmatori commettono errori, e non pochi, mentre scrivono programmi. Quando commettete un errore, sarebbe poco efficiente

dover ribattere le intere linee. Il BASIC ST dispone di un editor incorporato che vi permette di vedere un programma sullo schermo e di aggiungere, cancellare o modificare le informazioni all'interno del programma con i tasti freccia e i tasti funzionali.

Per sottoporre un programma ad editing, battete **EDIT** e premete Return. Se volete andare ad uno specifico numero di linea, potete far seguire alla parole EDIT il numero della linea in questione. Il vostro programma BASIC verrà visualizzato nella finestra Edit, entro la quale potrete spostarvi usando i tasti freccia. La Figura 2.4 mostra la finestra Edit. Per imparare come funziona il procedimento di editing, battete il seguente programma (nella finestra Command, come al solito), **esattamente** come si presenta qui, errori compresi; lo correggerete poi, con EDIT.

```
 5 REM Vogliamo sbarazzarci di questa linea
10 PRINT "Questo e' un test per il sistema di trasmissione d'emergenza."
30 REM Potremmo anche usare una linea 20
40 PRINT "Premi Return per finire il programma";A$
```

Questo programma presenta diversi problemi, quindi battete **EDIT** per andare alla finestra Edit. Provate a spostarvi nel programma con i tasti del cursore, poi ritornate nell'angolo in alto a sinistra, da dove eravate partiti. Userete i tasti funzionali per correggere il programma, troverete la funzione di cia-

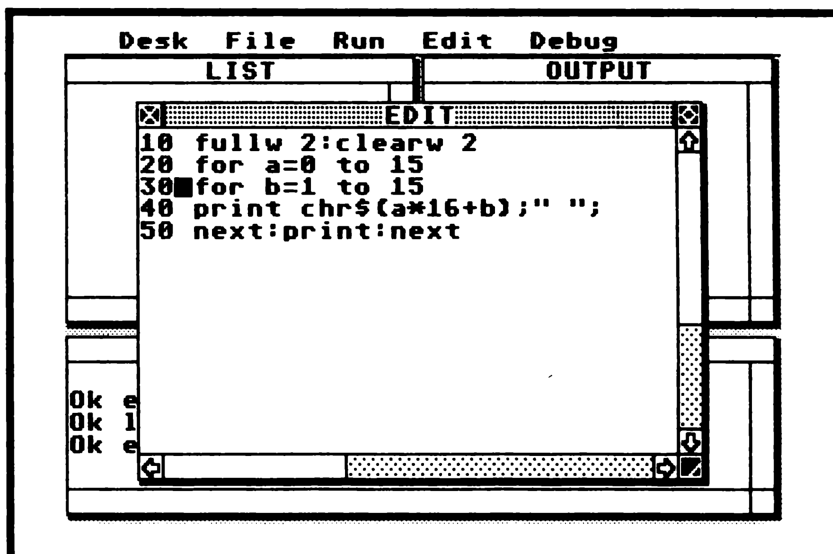


Figura 2.4: la finestra Edit.

scuno di essi nella Tabella 2.2.

Ora andate alla linea 5 e premete il tasto F4 per cancellare la linea. La linea 5 diventera *disalinneata*(sembrerà più luminosa del resto), che indica che è stata cancellata. Per farla sparire del tutto, premete il tasto freccia in alto per ritornarvi e premete F4 un'altra volta.

Ora dovete togliere la *e* di troppo in **teest**; spostate il cursore in modo che vada a trovarsi sovrapposto ad una della due **e** e premete F2 per cancellare un carattere. Di nuovo la linea 10 diventerà disallineata, ciò questa volta indica semplicemente che avete apportato qualche cambiamento. Premete Return per far sapere al computer che avete finito l'editing di quella linea; la linea tornerà al suo aspetto normale.

Ora il cursore dovrebbe trovarsi all'inizio della linea 30; poiché vogliamo inserire una linea in questo punto, premete F3 e le linee 30 e 40 dovrebbero entrambe spostarsi verso il basso di una linea. Battete la linea 20:

```
20 GOTO 40
```

Ora spostate il cursore alla linea 40 per poter cambiare l'istruzione PRINT in INPUT. Sovrapponete il cursore alla **P** di PRINT e battete direttamente la parola **INPUT.** Non dovrete inserire né cancellare nulla, perché la parola INPUT si sovrapporrà a PRINT. Premete Return e poi F10 per uscire dalla finestra Edit.

F1	Inserisce uno spazio all'attuale posizione del cursore
F2	Cancella un carattere nell'attuale posizione del cursore
F3	Inserisce una linea nell'attuale posizione del cursore
F4	Cancella una linea nell'attuale posizione del cursore
F5	Si sposta di una pagina verso l'alto nel programma
F6	Si sposta di una pagina verso il basso nel programma
F7	Carica il testo da disco
F8	Salva il testo su disco
F9	Cancella il buffer (area di memoria) di testo in uso
F10	Esce dalla modalità di editing, salvando i cambiamenti apportati

Tabella 2.2: i tasti funzionali usati nell'editing.

I comandi di editing dei tasti funzionali e della barra dei menù rendono agevole l'apportare cambiamenti al programma. Provate le funzioni di editing ancora per un pò per abituarvi ai diversi modi in cui potete cambiare linee e caratteri. State pur certi che una certa destrezza nell'editing vi tornerà utile quando inizierete a scrivere programmi più lunghi.

DELETE

Oltre alla finestra dell'editing, ci sono anche diverse parole chiave del BASIC ST che vi possono aiutare per ripulire o modificare il programma. Una di queste, DELETE, vi permette di eliminare alcune linee dal programma. Ci sono diversi modi di usare DELETE:

DELETE *numero*

cancella un numero di linea (esempio: DELETE 50)

DELETE *numero-numero*

cancella tutto dal primo numero di linea al secondo (esempio: DELETE 60-170)

DELETE *numero-*

cancella tutti i numeri di linea da quello specificato fino alla fine del programma (esempio: DELETE 230-)

DELETE *-numero*

Cancella il programma fino al numero di linea specificato compreso (esempio: DELETE -200).

Il comando LIST, per inciso, ha lo stesso formato del comando DELETE e può adottare le stesse variazioni. La differenza, naturalmente, sta nel fatto che LIST visualizza le linee specificate nella finestra List, anzichè cancellarle.

STEP

Un altro comando utile che vi può aiutare a trovare i bugs cioè gli errori del programma è STEP. Il formato di questo comando immediato è semplicemente:

STEP

Quando inserite STEP, il computer inizia a far girare il programma in memoria una linea per volta. Dopo ogni linea, il computer si ferma ed aspetta che premiate Return prima di passare alla successiva. Vedendo i risultati di ciascuna linea, potrete facilmente individuare le aree che richiedono qualche ritocco.

FOLLOW

FOLLOW è una specie di parola chiave, poiché segue i cambiamenti che si verificano nelle variabili selezionate nel corso del programma:

FOLLOW *variabile[,variabile,...].*

Mentre il programma sta girando, il computer visualizzerà sullo schermo ogni variabile selezionata che cambia, nel momento in cui cambia. In tal modo potete osservare certe variabili cambiare valore mentre il programma sta girando. Dovrete comunque specificare quali variabili volete visualizzare, quindi elencate le variabili (separate dalla virgola) che volete seguire con FOLLOW (per esempio, FOLLOW B,XX). Quando volete disinserire questa funzione, battete **UNFOLLOW.**

REM e RENUM

Un altro paio di modi per ripulire i programmi prevede le istruzioni REM e RENUM. REM dice al computer d'ignorare ogni cosa si trovi sulla linea che inizia con REM. REM è l'abbreviazione di **remark,** cioè commento e lo potrete usare tutte le volte che vorrete scrivere un'annotazione a vostro esclusivo uso e consumo (o eventualmente di un'altra persona che userà il programma) su quello che certe parti del programma fanno. A REM potete sostituire un apostrofo ('). Entrambe le linee del seguente esempio verranno ignorate dal computer, ma saranno leggibili all'utente quando il programma verrà listato:

```
50 REM Questa parte del programma calcola le funzioni
60 'Le funzioni vengono calcolate usando le formule della linea 200
```

RENUM rinumera il programma nel modo che voi specificherete. Spesso un programma finisce per avere una numerazione irregolare delle linee, dopo che il programmatore ne ha aggiunto, cancellato e modificato alcune. RENUM

può ridisporre le linee (e tutti i riferimenti alle linee) in modo più ordinato, per esempio seguendo il modello 10, 20, 30, 40 e così via. Il formato di questo comando è:

RENUM *linea nuova, numero della linea iniziale[,incremento]*

La *linea nuova* è quella che diventerà la prima linea nel programma rinumerato; il *numero della linea iniziale* è il punto in cui volete che il computer cominci a rinumerare; l'*incremento* indica quale passo il computer dovrà usare per rinumerare le linee. RENUM 50,100,20, quindi, rinumera tutte le linee da 100 alla fine del programma, in modo che i nuovi numeri di linea partano da 50 e procedano per incrementi di 20 (70, 90, 110 e così via). Battendo RENUM da solo, si rinumera tutto il programma, con numero iniziale 10 e i seguenti incrementati di 10 (default).

Altri comandi di editing

Vediamo una breve descrizione di altre parole chiave del BASIC che vi possono aiutare ad individuare e risolvere alcuni problemi del programma.

ERL La parola chiave ERL funziona come una variabile di ricerca, poiché in essa viene memorizzato il numero di linea dove si trova l'errore più recente. Quindi se avete un'istruzione per individuare gli errori, quale **ON ERROR GOTO 500** all'inizio di un programma e qualche errore fa andare il computer alla linea 500, potreste a quel punto emettere il comando PRINT ERL per vedere dove si è verificato l'errore.

ERR Analogamente, ERR registra il codice dell'errore più recente. L'Appendice B riporta l'elenco dei messaggi d'errore e dei rispettivi codici.

ERROR Se volete simulare un errore nel programma, battete ERROR seguito dal codice dell'errore che volete creare. Vedendo cosa accade nel programma quando si verifica un certo errore, potete rendere il programma più affidabile.

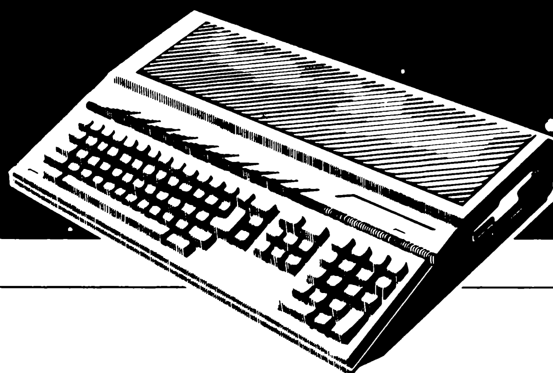
END L'istruzione END interrompe il programma immediatamente e vi riporta alla finestra Command. Spesso troverete opportuno creare un'interruzione al centro del programma per permettervi di controllare le variabili o altri importanti fattori man mano che il programma procede. Nel debugging, potete impostare diverse istruzioni END, per avere la possibilità di esaminare con attenzione quello che il programma sta facendo.

STOP Invece di arrestare il programma completamente, l'istruzione STOP vi pone nella modalità di riposo. Nella modalità di riposo, potete controllare la situazione delle variabili, dire al programma di continuare a girare con il comando CONT o ritornare alla modalità normale battendo **STOP** e premendo Return.

RESUME Se avete una routine per la gestione degli errori, potete mettere RESUME alla fine della routine per istruire il computer sulla sua prossima destinazione. Per esempio, potreste avere **RESUME 70,** che dice al computer "ora che il guaio è stato rimediato, va alla linea 70". **RESUME NEXT** continua il loop che stava eseguendo e **RESUME** rimanderà il programma nel punto in cui è incorso nel primo errore. Troverete esempi di gestione degli errori in tutto il libro.

VERSO UNA BUONA PROGRAMMAZIONE

Programmare non è facilissimo, ma una volta che avete acquisito gli elementi di base, può rivelarsi molto divertente. Assicuratevi di aver assimilato il contenuto di questo capitolo prima di procedere oltre. Una volta che vi sarete impadroniti di questo materiale, così ordinario ma fondamentale, potrete esplorare materie più interessanti, utili e divertenti nei capitoli che seguono.



PARLE E NUMERI

Le fondamenta dei programmi sono costituite dalle parole chiave matematiche e di testo di un linguaggio. Il BASIC ST presenta una grande varietà d'istruzioni matematiche e di stringa, che vi permettono di calcolare espressioni matematiche, manipolare stringhe e visualizzare sullo schermo caratteri per riprodurre i quali non basta la tastiera.

In questo capitolo concentrerò la mia attenzione su queste funzioni e istruzioni e sul loro uso.

COMANDI, ISTRUZIONI E FUNZIONI MATEMATICHE

Abbiamo già visto nel Capitolo 2 che cosa è una variabile numerica; tuttavia ci sono molte istruzioni del BASIC ST che rendono le variabili numeriche più versatili. Per la maggior parte queste parole chiave sono funzioni, ma alcune sono istruzioni o comandi.

Gestione della memoria

Ricorderete certamente che le variabili numeriche (designate con etichette del tipo YY, X3, TOTAL, VENDITE1987 ed altre combinazioni di lettere ed eventualmente di cifre) rappresentano dei numeri ed occupano una certa porzione di memoria. Più variabili avrete, specialmente se lavorerete su grandi matrici, più memoria richiederanno. Vediamo alcune parole chiave che vi aiuteranno a gestire meglio le vostre variabili e la memoria.

SWAP

L'istruzione SWAP scambia i valori di due variabili diverse:

SWAP *variabile1,variabile2*

Supponiamo che abbiate una variabile A uguale a 40 e una variabile B uguale a 650. Per assegnare il valore di A alla variabile B ed il valore di B alla variabile A, basta battere *SWAP A,B.*

Questo genere di scambio di valori di solito ricorre quando si devono ordinare dei numeri, e diversamente dal comando CLEAR, SWAP (come ERASE) richiede un argomento per operare. Un comando come CLEAR può farne a meno.

CLEAR

Il comando CLEAR *ripulisce* la memoria del computer. Tutte le variabili, le matrici, i valori dei loop FOR...NEXT e gli indirizzi dei dati vengono tutti eliminati dalla memoria. Il programma sarà ancora in memoria dopo che avrete battuto *CLEAR*, ma per il resto tutto sarà come se foste appena entrati nel BASIC ST ed aveste appena lanciato il programma attuale.

ERASE

Se volete essere più specifici su quali matrici volete eliminare dalla memoria, potete battere *ERASE* seguito dal nome della matrice (o delle matrici):

ERASE *nome della matrice[,nome della matrice,...]*

Per esempio, supponete di avere in memoria le matrici G(x), HH(x) e volervene sbarazzare. Basta battere *ERASE G,HH.* Questa istruzione è talvolta preferibile a CLEAR, in quanto può capitare di voler cancellare solo due variabili dalla memoria ma non tutti gli altri dati.

Istruzioni per la definizione delle dimensioni

Quando si scrive un programma, si ha un'idea approssimativa di quanto grande o piccola una variabile diventerà. La variabile che indica il numero dei giocatori di una certa partita per esempio, potrà essere 3 o 4, ma la variabile di una complessa formula matematica conterrà probabilmente un numero molto grande. Per non sprecare spazio nella memoria, il computer può assegnare una certa quantità di memoria a ciascuna variabile. Le variabili che conterranno numeri molto grandi richiederanno quantità di memoria maggiori rispetto alle variabili con numeri più piccoli. Le variabili numeriche piccole vengono anche definite *variabili intere*, quelle un po' più grandi si definiscono *a precisione singola* e quelle particolarmente grandi si dicono *a precisione doppia.* Un'intera occupa due byte di memoria, un numero a precisione singola occupa quattro byte e i numeri a precisione doppia occupano otto byte. Tenete presente che i numeri più precisi sono "grandi" solo nel senso che occupano una porzione più grande di memoria. Per esempio, anche se il valore 3,1415926 è decisamente minore di 1,000,000, la maggiore precisione del primo lo rende più grande del secondo in termini di memoria occupata.*

Potete gestire specifiche variabili definendo le dimensioni che volete che as-

* Nota: secondo la notazione anglosassone, adottata dall'informatica, il punto decimale sta per la nostra virgola e viceversa.

sumano. Per esempio, se volete usare variabili chiamate UU, VV e XX che conterranno solo interi, potreste risparmiare un po' di spazio battendo *DEFINT UU-XX* per comunicare al computer che le variabili UU, VV e XX vanno definite come intere.

Anche se potete aggiungere al nome delle variabili dei simboli che ne dichiarino il tipo (% per quelle intere, ! per quelle a precisione singola e # per quelle a precisione doppia), ricorrere ad una funzione di definizione è molto più efficace quando volete assegnare il tipo a gruppi di tre o più nomi di variabile. Quella che segue è una lista completa delle istruzioni di definizione delle dimensioni.

DEFINT

DEFINT definisce un gruppo di nomi di variabile come intere:

DEFINT *lettera-lettera*

Nel BASIC ST i numeri sono compresi nell'intervallo —32767 e 32768 (per esempio 50.3 non è un intero, ma 50 sì). Aggiungete a DEFINT un intervallo di variabili (dicendo in tal modo al computer che tutte le variabili comprese tra la prima e la seconda saranno intere).

Notate che la dimensione minima di un gruppo di variabili può anche essere una variabile sola. DEFINT X-Y rende X,Y e tutte le variabili che iniziano con X o Y variabili intere. Poiché tutte le variabili che iniziano con una qualsiasi delle lettere di un certo intervallo specificato saranno considerate variabili intere, se battete, per dire, *DEFINT H-K*, la variabile JJ1 sarà una variabile intera poiché la sua prima lettera, J, è compresa nell'intervallo di variabili che abbiamo definito intere.

DEFSNG

DEFSNG definisce gruppi di variabili a precisione singola, che hanno una accuratezza di sei cifre significative:

DEFSNG *lettera-lettera*

Le variabili vengono automaticamente definite a precisione singola, quindi probabilmente non dovrete ricorrere a questa funzione. Se volete proprio assegnare la precisione singola alle variabili, battete *DEFSNG* seguito dall'intervallo.

DEFDBL

Se pensate di usare numeri molto grandi o molto precisi, troverete opportuno assegnare la precisione doppia a certi intervalli di variabili, con un'accuratezza di nove cifre significative. Battete *DEFDBL* seguito dal gruppo di variabili in questione:

DEFDBL *lettera-lettera*

Questa istruzione impiega più memoria di DEFSNG, ma vi permette di aumentare sensibilmente le dimensioni e l'accuratezza delle variabili specificate.

DEFSTR

Le variabili di stringa sono di solito seguite dal simbolo del dollaro, quindi RR$, BC$ e DL$ rappresentano tutte variabili di stringa. Se volete rappresentare le stringhe senza ricorrere al simbolo del dollaro, potete usare l'istruzione DEFSTR:

DEFSTR *lettera-lettera*

DEFSTR stabilisce che tutte le variabili il cui nome inizia con una certa lettera sono variabili di stringa. Per esempio, se battete *DEFSTR A-C*, tutte le variabili che cominciano con A, B o C saranno variabili di stringa e non le potrete usare come variabili numeriche: *Alpha=9* non sarà consentito.

Funzioni di ridefinizione dei numeri

Oltre ad essere in grado di risparmiare memoria o di riservare più ampie quantità di memoria a certe variabili, potrete cambiare le caratteristiche dei numeri o delle variabili con le seguenti funzioni:

STR$

Se volete mettere un numero in una variabile di stringa, usate la funzione STR$ con il numero come argomento:

variabile di stringa =STR$(*espressione numerica*)

Per esempio, se la variabile YY contiene il numero 47.2 e voi battete *A$=STR$(YY)*, la variabile di stringa A$ sarà uguale a 47.2. STR$ trasforma automaticamente il primo carattere di questo tipo di variabile in uno spazio vuoto se il numero è positivo e in un segno meno se il numero è negativo. Alcuni risultati di STR$ potrebbero essere i seguenti:

—50
22.98
—70

Troverete molto utile porre dei numeri nelle stringhe quando dovrete formattare i numeri sullo schermo in un certo modo.

VAL

L'opposto di STR$ è la funzione VAL, che trasforma una stringa di cifre in un numero che può essere memorizzato in una variabile numerica:

variabile numerica =VAL*(stringa di cifre)*

Se una stringa è uguale ad un numero, quel numero potrà essere stampato, memorizzato in una stringa o usato in qualsiasi modo voi specifichiate. Se H$ è uguale a 506, *PRINT VAL(H$)* produrrà il numero 506 nella finestra Output e *H=VAL(H$)* memorizzerà il numero 506 nella variabile H. Se la stringa contiene più di un numero, VAL estrarrà il numero nella stringa solo se ne costituisce la prima parte; *PRINT VAL("21 anni e' la mia eta' ")* come risultato stamperà il numero 21 nella finestra Output e *PRINT("La mia eta' e' di 21 anni")* produrrà uno 0, dato che il numero non occupa l'inizio della stringa. Usando VAL con una stringa che non contiene numeri, allo stesso modo si otterrà uno 0.

ABS

In matematica la funzione di valore assoluto è semplice: elimina il segno meno in un numero negativo e lascia intatti i numeri positivi. Analogamente, la funzione ABS del BASIC ST restituirà il valore positivo di qualsiasi numero dato:

variabile numerica =ABS*(espressione numerica)*

Vediamo alcuni esempi:

```
PRINT ABS(50)
50
PRINT ABS(—60)
60
PRINT ABS(0)
0
```

SGN

Un'altra funzione che ha a che fare con i segni più e meno è SGN, che determina se una variabile è positiva, negativa o è uguale a zero:

variabile numerica = SGN*(espressione numerica)*

Notate che SGN è seguito da un numero o da una variabile tra parentesi e produrrà 1 (per indicare positivo) —1 (per indicare negativo) e 0 (che significa che il numero o la variabile sono uguali a zero). *PRINT SGN(—50), SGN(1), SGN(0),* darà come risultato nella finestra Output —1, 1 e 0.

FIX e INT

La funzione FIX elimina la parte decimale di un numero:

variabile numerica = FIX*(espressione numerica)*

PRINT FIX(50.5) darà come risultato 50, e *X=FIX(4.9999),* porrà X uguale a 4.

La funzione INT fa esattamente la stessa cosa usando lo stesso formato:

variabile numerica = INT*(espressione numerica)*

PRINT INT(—40.40923) farà comparire -40 nella finestra Output.

MOD

Quando eseguite delle divisioni, avrete spesso un resto: 70/6, per esempio, risulta 11.6666666. La funzione MOD:

variabile numerica=*numero* MOD *divisore*

vi permette di trovare il resto di una divisione, in modo da evitare lunghi numeri decimali. *PRINT 70 MOD 6* vi dice il resto dell'operazione 70/6, che è 4; 70 diviso 6, quindi, è uguale a 11 con il resto di 4.

CINT

La funzione CINT:

variabile intera =CINT(*espressione numerica)*

è diversa da INT perché invece di eliminare la parte decimale del numero, lo arrotonda. Se la parte decimale di un numero è minore di 0.5, la funzione CINT elimina tutta la parte che segue la virgola e lascia intatto il numero intero; invece se la parte decimale è maggiore di 0.5, la funzione CINT arrotonda il numero al successivo numero intero. *PRINT CINT (4.49)* vi restituisce il valore 4, mentre *PRINT CINT (60.7)* vi restituisce il valore 61.

Funzioni di conversione dei numeri

Altre due funzioni, HEX$ e OCT$, convertono i numeri da una base ad un'altra. I loro formati sono:

variabile numerica =HEX$(*espressione numerica)*
variabile numerica =OCT$(*espressione numerica)*

Come probabilmente già sapete, il nostro sistema di numerazione è in base 10 (ed impiega le cifre da 0 a 9). Il sistema esadecimale (in base 16), invece, usa i numeri da 0 a 9 e le lettere da A ad F. Alcuni numeri esadecimali potrebbero essere AFFF, 9E e 1000. Per convertire un numero decimale in una stringa esadecimale, si usa la funzione HEX$ seguita dal valore numerico tra parentesi. Vediamo un paio di esempi:

```
PRINT HEX$(254)
FE
PRINT HEX$(65535)
FFFF
```

I numeri esadecimali sono importanti nella programmazione avanzata, specialmente se si tratta di linguaggio assemblatore.

Talvolta assumono qualche rilievo i numeri ottali (in base 8). Potrete convertire i numeri decimali in numeri ottali con lo stesso formato del comando HEX$; per la conversione in ottali si ricorre alla funzione OCT$ seguita dal numero tra parentesi.

Funzioni matematiche

Le variabili matematiche di cui abbiamo parlato finora in qualche modo manipolano le variabili.

Il BASIC ST presenta anche diverse funzioni matematiche che lavorano direttamente su espressioni numeriche e che potete usare come parti costitutive in equazioni più complicate. Le funzioni che seguono possono agire da sole o in combinazione tra loro.

SQR

La funzione SQR calcola la radice quadrata di un numero o variabile:

variabile numerica =SQR*(espressione numerica)*

EXP

EXP, la funzione esponenziale, eleva il numero che specificate alla potenza della costante matematica *e* (circa 2.71828):

variabile numerica =EXP*(espressione numerica)*

LOG

La funzione LOG vi darà il logaritmo naturale di un numero:

variabile numerica =LOG*(espressione numerica)*

Questa funzione è l'inverso di EXP.

LOG10

Per trovare il logaritmo in base 10 di un numero o variabile, si usa LOG10:

variabile numerica =LOG10*(espressione numerica)*

SIN

Il BASIC ST dispone di molte funzioni trigonometriche e SIN è la prima e fondamentale di esse. SIN calcola il seno dell'angolo specificato e il suo formato è:

variabile numerica =SIN*(espressione numerica)*

Il seno di un angolo è uguale alla lunghezza del lato opposto divisa per l'ipotenusa di un triangolo rettangolo e l'angolo che date come argomento *espressione numerica* deve essere espresso in radianti.

Se conoscete solo l'angolo, dividetelo per 57.29578 ($180/\pi$) per trovare il valore in radianti; una volta che ne conoscete il risultato, inseritelo tra parentesi dopo SIN.

COS

La funzione COS calcola il coseno dell'angolo espresso in radianti:

variabile numerica =COS*(espressione numerica)*

Il coseno di un angolo è la lunghezza del lato adiacente diviso per la lunghezza dell'ipotenusa.

TAN

TAN calcola la tangente dell'angolo da voi specificato (come al solito, l'angolo deve essere misurato in radianti):

variabile numerica =TAN*(espressione numerica)*

La tangente è uguale al seno diviso per il coseno, ovvero la lunghezza del lato opposto diviso per la lunghezza del lato adiacente.

ATN

ATN calcola l'arcotangente dell'angolo da voi specificato:

variabile numerica =ATN*(espressione numerica)*

L'arcotangente è l'inverso della tangente.

Come si definiscono le funzioni speciali

Se pensate di dover ricorrere frequentemente ad una certa operazione che non è predefinita come funzione del BASIC ST, potrete trovare opportuno definire tale funzione con l'istruzione DEF FN. DEF FN imposta una definizione e le assegna un nome. Se doveste per esempio usare spesso la formula X=(A+B*5.27*B*B), dovreste battere *DEF FNMIA(A,B)=(A+B*5.27*B*B).* Dopo aver inserito questa definizione di funzione, potrete trovare il risultato della formula per qualsiasi valore dato ad A e B semplicemente battendo *PRINT FNMIA(A,B),* dove A e B sono i due numeri; *PRINT FNMIA(5,4),* per esempio, risulta 342.28. Il formato di DEF FN è:

DEF FN*NOME DELLA FUNZIONE (variabile[,variabile,...])=espressione*

Il *NOME DELLA FUNZIONE* (scritto a lettere maiuscole) è il nome che volete attribuire alla funzione, le *variabili* sono le variabili numeriche impiegate nella funzione e l'*espressione* è la formula che volete definire. Tutte le volte che volete usare la funzione, battete *FN* seguita (senza lasciare spazi) dal nome della variabile e poi dalle variabili che dovete usare tra parentesi.

Nel prossimo esempio, definisco la funzione CUB per calcolare il cubo di un qualsiasi numero. L'istruzione *DEF FNCUB(N)=(N*N*N)* della linea 20 dice al computer che CUB deve prendere qualsiasi numero specificato e moltiplicarlo per il numero stesso tre volte; quando date al computer un numero, ne calcolerà il cubo semplicemente dandogli il comando FNCUB.

```
 5 REM Programma di elevamento al cubo
10 FULLW 2: CLEARW 2
20 DEF FNCUB(N)=(N*N*N)
30 INPUT "Inserisci il numero";N
40 PRINT "Il cubo e' ";FNCUB(N)
50 GOTO 30
```

RND e RANDOMIZE

Le ultime due parole chiave relative ai numeri sono RND e RANDOMIZE. La funzione RND, che incontrerete spesso nel capitolo sulla grafica e in quello della sintesi acustica (anche se i numeri casuali si possono trovare in qualsiasi tipo di programma), genera un numero casuale compreso tra 0 e 1 esclusi, vale a dire un numero frazionario tra 0 e 1, che non comprende 0 e 1. Battendo *PRINT RND* può risultare 0.23432, 0.78564 o 0.50232. Tuttavia i numeri casuali decimali non sempre sono utili (per esempio, quando volete generare suoni casuali e vi servono numeri alti). Quindi vediamo una formula molto semplice per ottenere numeri casuali compresi tra 1 ed un altro numero da voi specificato:

```
X=INT(numeroxRND)+1
```

Il numero che specificate verrà moltiplicato per un numero casuale, la parte decimale verrà eliminata e verrà aggiunto 1 per arrotondare il numero casuale. Quindi, se vi serve un numero casuale tra 1 e 400, batterete *X=INT(400*RND)+1.* Il valore di numeri generati casualmente diventerà più evidente quando scriverete programmi che prevedono la casualità nel Capitolo 4.

L'istruzione RANDOMIZE persegue uno scopo complementare a quello della funzione RND, poiché reimposta il generatore di numeri casuali del computer. I numeri prodotti dal computer hanno un grado abbastanza alto di casualità, ma non sono veramente casuali; usando RANDOMIZE, potete porre un altro numero *seme* nel generatore di numeri casuali per aumentare il grado di casualità del risultato della funzione RND. RANDOMIZE non è d'importanza fondamentale, ma potete usarla se volete conferire la maggiore casualità possibile ai numeri generati dal computer.

L'INSIEME DEI CARATTERI ASCII

Per permettere a diversi prodotti di funzionare insieme, esistono diversi standard a cui i diversi produttori aderiscono.

Uno di questi si chiama *insieme di caratteri ASCII* (American Standard Code for Information Interchange), che non è altro che un sistema di 256 numeri che corrispondono a 256 caratteri. L'insieme dei caratteri ASCII può subire lievi variazioni da un computer ad un altro, poiché le varie funzioni dei computer non sono identiche, ma molti codici di caratteri ASCII (e principalmente

quelli che riguardano le lettere, i numeri e i simboli più comuni) sono esattamente gli stessi.

Il sistema ASCII è piuttosto semplice: ogni numero da 0 a 255 corrisponde ad un carattere.

Il codice ASCII 65 rappresenta la A maiuscola e il codice 32 rappresenta uno spazio vuoto (" "). In effetti il codice ASCII 65 rappresenta la A maiuscola sia che lo verifichiate su Atari, Amiga, Commodore 64, TRS-80, Apple, PC IBM e un altro computer.

Lo scopo principale di questo insieme di caratteri è quello di permettere ai computer di comunicare tra loro senza dover tradurre.

ASC

Una delle due funzioni che hanno direttamente a che fare con il set ASCII è ASC:

variabile intera =ASC(*espressione di stringa*)

Usando ASC, potete trovare il codice ASCII di qualsiasi *espressione di stringa* Quindi:

```
PRINT ASC ("C")
```

restituisce il numero 67, dato che il codice ASCII per la C maiuscola è proprio 67. Se battete:

```
H=ASC("E")
```

verrà registrato il numero 69 nella variabile H. Il programma seguente vi permette d'inserire un carattere in modo che il computer ne possa trovare il valore ASCII:

```
 5 REM Trova il codice ASCII
10 FULLW 2: CLEARW 2
20 INPUT "Inserisci il carattere";A$
30 PRINT "Il codice ASCII per"
40 PRINT A$;" e' ";ASC(A$)
50 GOTO 20
```

Se l'espressione di stringa è più lunga di un carattere, il computer dà il codice solo del primo carattere.

CHR$

L'inverso della funzione ASC è CHR$, che prende un codice ASCII *(espressione numerica)* e visualizza il carattere corrispondente *(variabile di stringa):*

variabile di stringa =CHR$*(espressione numerica)*

Proprio come *PRINT ASC("A")* dà il valore 65, *PRINT CHR$(65)* visualizza sullo schermo la A maiuscola. Ricordate che l'argomento deve essere compreso tra 0 e 255. Inoltre l'insieme dei caratteri ASCII contiene molto più che le sole lettere, numeri e simboli; nel set sono inclusi anche interessanti segni grafici, un segnale acustico ed altre utili funzioni, che sono riportati nell'Appendice D. Vediamo un breve programma che visualizza sullo schermo l'intero set di caratteri:

```
 5 REM Visualizza l'insieme di caratteri ASCII
10 FULLW 2: CLEARW 2
20 FOR A=0 TO 15
30 FOR B=1 TO 15
40 PRINT CHR$(A*16+B);" ";
50 NEXT: PRINT: NEXT
```

Funzioni di formattazione

Quando dovete visualizzare un testo sullo schermo, può darsi che lo vogliate vedere in qualche formato particolare. Per esempio, se avete battuto cinque colonne di numeri, è probabile che vorrete assicurarvi che le colonne siano allineate e i numeri si presentino in modo ordinato. Il BASIC ST prevede diverse funzioni che vi permettono di stampare caratteri o stringhe in determinati punti dello schermo in modo che possiate dare al vostro testo il formato che desiderate.

TAB

La prima di queste funzioni, TAB, vi permette di stampare qualsiasi cosa spostata verso destra di un numero prefissato di spazi rispetto al margine sinistro dello schermo:

PRINT TAB*(posizione)[“stringa”]*

PRINT TAB(5) "Hello", per esempio, stamperà la parola *Hello* spostata di cinque spazi verso destra. I valori del TAB del computer, come quelli di una macchina per scrivere, sono posizioni assolute; vale a dire che TAB (10) è sempre localizzato a dieci spazi verso destra dal bordo sinistro dello schermo. Battendo:

```
PRINT TAB (20) "Prova" TAB (20) "Prova"
```

si otterrà:

Prova
Prova

Siccome TAB (20) è una posizione ben precisa, e siccome il computer non userà lo stesso punto due volte, salterà alla linea successiva in modo da poter raggiungere la posizione TAB (20) e stampare la seconda *Prova.*

SPC

SPC, al contrario, riguarda le posizioni relative, anziché assolute. Esso infatti stampa semplicemente un certo numero di spazi in modo che il carattere successivo verrà stampato in un determinata posizione verso destra rispetto all'ultimo carattere stampato:

PRINT SPC*(espressione numerica)*

Nel precedente esempio relativo a TAB, le due *Prova* si trovavano su due linee diverse, dato che TAB aveva dovuto saltare alla linea successiva per poter raggiungere una posizione di TAB 20 che non fosse già occupata. SPC, invece, si limita a stampare un numero di spazi da voi specificato. Una linea di programma del tipo

```
PRINT SPC(10) "Prova" SPC(10) "Prova"
```

farà comparire quanto segue nella finestra Output:

Prova   Prova

Il computer non deve più saltare alla linea successiva; si sposta verso destra di dieci spazi, stampa *Prova,* si sposta di altri dieci spazi, poi stampa la seconda *Prova.*

SPACE$

Un'altra parola chiave, SPACE$, può svolgere una funzione analoga; il risultato che abbiamo appena visto potrebbe anche essere conseguito battendo:

PRINT SPACE$ (10) "Prova" SPACE$ (10) "Prova"

La differenza sta nel fatto che potete usare SPACE$ anche per creare o aggiungere ad una stringa. Battendo qualcosa di simile a :

A$="Questa e' una"+SPACE$(15)+"lunga stringa."

A$ sarà uguale a:

Questa e' una lunga stringa.

La funzione SPACE$ può far parte di una stringa o costituire una stringa di per sé. Il suo formato è:

variabile di stringa =SPACE$*(espressione numerica)*

POS(0)

Se volete scoprire l'attuale posizione del cursore, potete ricorrere alla funzione POS(0); lo (0) è fittizio, vale a dire che non è significativo per la funzione, ma che è lo stesso importante inserirlo. POS è particolarmente utile in un programma quando state formattando e dovete controllare quale sia la posizione del cursore per poter collocare i dati successivi nella posizione appropriata. Il suo formato è:

valore intero =POS(0)

GOTOXY

Un metodo più diretto di posizionare il cursore è l'istruzione GOTOXY:

GOTOXY *colonna,riga*

GOTOXY invia il cursore in una certa parte dello schermo, specificata dai valori di X e Y che seguono l'istruzione. Il valore X (*colonna*) è la posizione

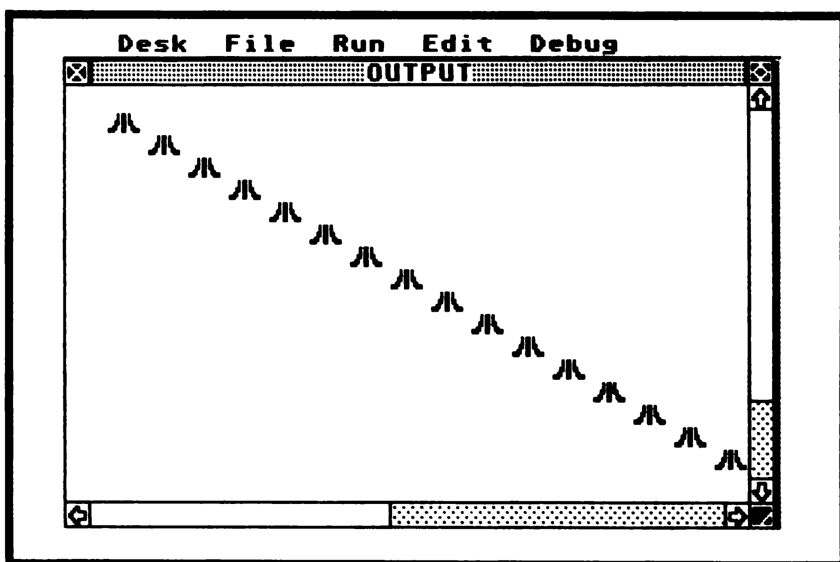


Figura 3.1: simbolo Atari disegnato con l'uso di GOTOXY e CHR$.

misurata in senso orizzontale del cursore sullo schermo, mentre il valore Y (*riga*) è la posizione misurata in senso verticale. *GOTOXY 5,7* porta il cursore 5 spazi a destra e 7 righe in basso rispetto all'angolo superiore sinistro dello schermo.

Esempi di programmi

Il prossimo programma impiega le funzioni GOTOXY e CHR$ per stampare 16 simboli dell'Atari in diagonale sullo schermo, come mostra la Figura 3.1.

```
 5 REM Simboli Atari con l'uso di GOTOXY e CHR$
10 FULLW 2: CLEARW 2
20 FOR P=1 TO 16
30 GOTOXY P*2,P
40 PRINT CHR$(14);CHR$(15);
50 NEXT
60 GOTO 60
```

Il secondo programma che usa anch'esso GOTOXY e CHR$ è un po' più interessante. Per qualche strana ragione, i programmatori della Atari che dovevano decidere il set di caratteri per l'ST hanno fissato quattro caratteri (dal

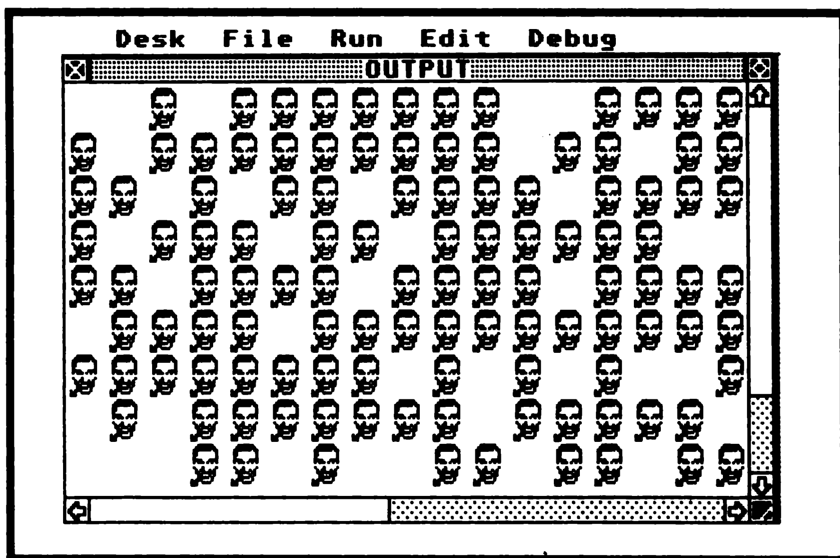


Figura 3.2: la Chiesa del Sub-Genio

28 al 31) che, se usati contemporaneamente, tracciano sullo schermo la faccia di un uomo. Questa faccia sorridente, che fuma la pipa, si chiama Bob ed è oggetto di un culto umoristico, "la chiesa del Sub-Genio". Il programma seguente stampa la faccia di Bob in punti casuali dello schermo, come mostra la Figura 3.2.

```
 5 REM La chiesa del Sub-Genio
10 FULLW 2: CLEARW 2: GOTOXY 3,0
20 PRINT "Bob all'attacco!"
30 FOR D=1 TO 900: NEXT: CLEARW 2
40 X=INT(RND*17)*2: Y=INT(RND*9)*2
50 GOTOXY X,Y
60 PRINT CHR$(28)+CHR$(29);
70 GOTOXY X,Y+1
80 PRINT CHR$(30)+CHR$(31);
90 GOTO 40
```

Notate il comando GOTOXY della linea 70 che impiega un valore Y che è maggiore di un precedente comando GOTOXY; ciò mette la parte inferiore della faccia di Bob sulla linea sottostante in corrispondenza della parte superiore.

STRING$

Anche il programma seguente impiega due linee, la linea superiore 12 visualizza 30 cifre identiche e conta da 0 a 9 e quella inferiore conta in verso opposto, da 9 a 0. Queste cifre sono caratteri ASCII che hanno l'aspetto di numeri visualizzati su una calcolatrice ed un nuovo comando, STRING$, rende facile stampare i 30 caratteri. Il formato di questo comando è:

STRING$*(numero,carattere)*

dove *numero* è il numero di caratteri che volete nella stringa e *carattere* è o il codice ASCII del carattere o il carattere stesso tra virgolette.

```
 5 REM Cifre
10 FULLW 2: CLEARW 2
20 FOR D=16 TO 25
30 GOTOXY 0,0
40 PRINT STRING$(30,D)
50 PRINT STRING$(30,41—D)
60 NEXT
70 GOTO 20
```

LAVORARE CON LE STRINGHE

L'istruzione PRINT, come abbiamo già visto nel Capitolo 2, è una delle parole chiave più spesso usate nel BASIC ST. Una variazione di PRINT lo rende ancora più versatile: *PRINT USING.* Questa parola chiave non solo stampa i numeri o le variabili sullo schermo, ma lo fa nel formato da voi specificato. Ciò è utile nel caso di dati scientifici, di cifre in dollari ed di altri numeri che devono essere stampati in qualche modo particolare. PRINT USING si usa esattamente come PRINT e la sua sintassi è:

PRINT USING *stringa,variabile o numero[,variabile o numero,...]*

La *stringa* dice a PRINT USING come formattare le variabili o i numeri. I *numeri* (o *variabili*) sono quello che volete stampare. Per esempio $$# # # è una stringa di formato che dice al computer di stampare i numeri preceduti dal simbolo del dollaro; se voleste stampare una variabile A, uguale a 354, la linea che dovreste battere ed il rispettivo risultato sarebbero:

```
PRINT USING "$$###";A
$354
```

Stringhe di formato

Ci sono parecchi tipi diversi di stringhe di formato che potete usare, ciascuna delle quali può essere combinata con altre. Vediamo un elenco delle diverse stringhe di formato e delle loro funzioni.

SIGN di numero

Il simbolo # specifica quante cifre verranno stampate in un numero. Usate tanti di questi segni quante saranno le cifre del numero da visualizzare. Se, per esempio, prevedete numeri con cinque cifre, userete #####.

SIGN dei decimali

Userete il segno dei decimali (.), cioè un punto nella notazione anglosassone, inserito tra due simboli di numero nella stringa di formato, se volete fissare un certo numero di cifre decimali. Per esempio,

```
PRINT USING "###.##"
```

dirà al computer di stampare tre cifre prima del segno di decimale e due dopo. Se il numero fosse 54.6743, il computer stamperà 54.67, eliminando le sue cifre in più.

Segno di dollaro

Per le cifre in dollari, si fanno precedere i segni di numero con due segni di dollaro ($$). I segni di dollaro nelle stringhe di formato faranno stampare un segno di dollaro davanti a qualsiasi numero visualizzerete con PRINT USING. Quindi,

```
PRINT USING "$$###.##";53.20
```

farà comparire nella finestra Output $53.20.

Segno positivo

Se volete che venga stampato il segno di un numero per indicare se è positivo o negativo, il primo carattere della stringa deve essere un più (+).

Asterischi

Quando usate PRINT USING con i segni di numero (#), il computer mette automaticamente degli spazi davanti a ciascun numero in modo che esso venga opportunamente allineato a destra. Se al posto degli spazi preferite che vengano stampati degli asterischi (**), mettete due asterischi nella stringa di formato. I numeri continueranno ad allinearsi in modo corretto, ma verranno visualizzati per esempio così:

```
****50.23**1250.65
```

Carattere di sottolineatura

Se c'è un carattere speciale che volete stampare con il numero, per esempio @, battete un carattere di sottolineatura (—) seguito dal carattere.

Quattro simboli di esponente

Includendo nella stringa di formato quattro simboli di esponente (^^^^) farete in modo che il computer stampi i numeri in notazione scientifica. Il numero 4320.5 comparirà nella forma 4.32E+03.

Punto esclamativo

Per quanto riguarda le stringhe, un punto esclamativo (!) nella stringa di formato farà in modo che il programma stampi solo il primo carattere della stringa da stampare.

Segno di e commerciale (&)

Il segno di e commerciale (&) specifica una stringa di lunghezza variabile (notate che l'intera stringa può essere stampata ricorrendo all'istruzione PRINT, senza alcuna stringa di formato).

Due barre rovesce

Userete due barre rovesce (\ \) quando vorrete stampare un certo numero di caratteri di una stringa più due. Una barra rovescia, tre (oppure *n*) spazi ed un'altra barra rovescia diranno al computer di stampare i primi cinque (o $n+2$) caratteri della stringa che avete battuto con PRINT USING. Per esempio, se A$ fosse MISSISSIPPI, e batteste

```
PRINT USING "\  \";A$
```

il computer visualizzerebbe *MISS* nella finestra Output. Le due barre rovesce e i due spazi vuoti hanno detto al computer di stampare i primi quattro caratteri della stringa.

Come si selezionano parti di stringhe

Alcune funzioni sono usate solo per selezionare parti di stringhe.

RIGHT$

RIGHT$, seguito dal nome della *stringa madre* e dal *numero di caratteri* che volete selezionare tra parentesi, estrae un certo numero di caratteri dall'estremità destra di una stringa madre:

stringa = RIGHT$(*stringa madre, numero di caratteri*)

RIGHT$ non altera in alcun modo la stringa madre, ma potrete usare la nuova *stringa* nel modo che più vi aggrada; la potete stampare, memorizzarla in un'altra stringa, trovare i suoi codici ASCII e così via.

Se, per esempio, volete lavorare con gli ultimi cinque caratteri più a destra di una stringa chiamata GG$ (che corrisponde a "Mi chiamo Carlo"), dovete battere:

```
PRINT RIGHT$(GG$,5)
```

per avere il risultante *Carlo* nella finestra Output.

Il programma seguente accetta dalla tastiera dati su città, stati e codici postali e, selezionando le ultime cinque cifre a destra, può visualizzare i codici postali da soli:

```
 5 REM Stampa Codici Postali
10 FULLW 2: CLEARW 2
20 LINE INPUT "Inserisci Citta', Stato e CAP:";ADR$
30 PRINT " Il CAP richiesto e' ";RIGHT$(ADR$,5)
40 GOTO 20
RUN
```

Inserisci Citta', Stato e CAP: Santa Clara, CA 95050
Il CAP richiesto e' 95050
Inserisci Citta', Stato e CAP:

LEFT$

LEFT$ funziona esattamente come RIGHT$, tranne che utilizza i caratteri all'estrema sinistra della stringa:

stringa = LEFT$*(stringa madre, numero di caratteri)*

Dalla stringa GG$ "Mi chiamo Sam", *PRINT LEFT$(GG$,2)* estrarrebbe **Mi** che comparirebbe nella finestra Output.

LEN

Un'altra funzione di stringa, LEN, restituisce il numero dei caratteri dell'argomento di una stringa:

numero intero = LEN*(espressione di stringa)*

Se A$ fosse uguale a "Io sono una stringa", *PRINT LEN(A$)* renderebbe il valore 19 e *PRINT LEN("Test")* restituisce un 4. LEN conta tutti i caratteri dell'argomento di una stringa, compresi gli spazi ed i caratteri speciali.

Il programma seguente impiega sia LEFT$ che LEN; dopo aver battuto una stringa, il programma usa un loop di tipo FOR...NEXT per contare all'indietro dalla lunghezza della stringa con passo 1. Per ciascun passo del loop, il computer stampa il numero di caratteri sul lato sinistro della stringa uguale al valore corrente nel contatore del loop. Ciò crea un interessante triangolo, poiché qualsiasi stringa inseriate, si accorcerà di un carattere e verrà visualizzata sulla linea successiva ad ogni passo del loop. Della stringa alla fine rimane solo l'ultimo carattere, che è l'ultima cosa visualizzata prima che il programma finisca, come potrete vedere osservando il programma mentre gira.

```
 5 REM La stringa che scompare
10 FULLW 2: CLEARW 2
20 INPUT "Inserisci una stringa";ST$
30 FOR I=LEN(ST$) TO 1 STEP —1
40 PRINT LEFT$(ST$,I)
50 NEXT
60 GOTO 60
RUN
```

Inserisci una stringa? Provare
Provare
Provar
Prova
Prov
Pro
Pr
P

MID$

Una funzione in qualche modo simile a RIGHT$ e LEFT$, ma più versatile, è MID$. MID$ può essere usata in due modi:

variabile di stringa = MID$*(stringa,punto iniziale,lunghezza)*
MID$*(stringa,punto iniziale,lunghezza)* = *variabile di stringa*

Nel primo modo estrae parte di una *stringa* a cominciare da una certo punto della stringa - *punto iniziale* - ed estrae un certo numero di caratteri - *lunghezza*. Quindi se volete estrarre cinque caratteri da H$ a partire dal sesto carattere e memorizzarli in una sottostringa Y$, basta battere:

```
Y$=MID$(H$,6,5)
```

Con la funzione MID$ si può anche sostituire una parte di stringa. Per esempio, per cambiare R$ in modo che il suo terzo, quarto e quinto carattere siano "IOU", battete:

```
MID$(R$,3,3)="IOU"
```

Il programma seguente impiega la funzione MID$ nella seconda forma. Dopo aver inserito la parola che il computer richiede, essa viene inclusa in una stringa. Dopo che la stringa viene stampata, la porzione di stringa contenente la vo-

stra parola viene sostituita dalla parola *dare*, come si può riscontrare nell'esempio. Provate il programma per vedere come funziona e notate come la funzione LEN viene impiegata nella linea 30 per accertare che la stringa inserita sia lunga quattro caratteri.

```
 5 REM Uso di MID$ per estrarre o sostituire parti di una stringa
10 FULLW 2: CLEARW 2
20 INPUT "Inserisci una parola di quattro lettere";WRD$
30 IF LEN(WRD$)<>4 THEN GOTO 20
40 TXT$="La parola di quattro lettere e' "+WRD$+", il che "+CHR$
   (10)+" non e' una sorpresa."
50 PRINT TXT$
60 MID$(TXT$,32,4)="dare"
70 PRINT TXT$
80 GOTO 80
RUN
```

Inserisci una parola di quattro lettere? Sic!
La parola di quattro lettere è Sic!, il che
non è una sorpresa.
La parola di quattro lettere è dare, il che
non è una sorpresa.

INSTR

La funzione INSTR può spesso essere usata con MID$, dato che il suo compito è quello di scoprire dove si trova un segmento di stringa. Il formato di INSTR è:

INSTR*([carattere iniziale,]stringa da cercare, numero di caratteri su cui eseguire la ricerca*

Se volete cercare il segmento di stringa *una* nella variabile SNT$ iniziando la ricerca dal quarto carattere della stringa, batterete *INSTR(4,SNT$,"una")*. L'uso che farete del valore restituito da INSTR dipende interamente da voi; potreste stamparlo, collocarlo in una variabile o farne qualsiasi altra cosa. La cosa importante è che il valore ottenuto rappresenta la posizione della stringa dove inizia il segmento da voi cercato. Nell'esempio precedente, se SNT$ è "Io ho una macchina" il valore reso sarà 7 poiché "una" inizia nella posizione del settimo carattere della stringa; quando usate INSTR non è necessario che specifichiate il carattere iniziale; se lo tralasciate, il computer eseguirà la ricerca del segmento da voi specificato in tutta la stringa; se non trova il segmento in questione, il valore che restituisce è 0.

Il prossimo programma impiega INSTR per trovare in quale città abita una persona. Dopo aver inserito la città, lo stato ed il codice postale, il computer cerca dove si trova il segmento CH (supponendo che tutti gli indirizzi inseriti siano della Svizzera). Una volta che il computer abbia trovato tale informazione, può estrarre il nome della città dalla stringa e visualizzarlo.

```
 5 REM Trovare il nome della citta'
10 FULLW 2: CLEARW 2
20 LINE INPUT "Inserisci citta' stato e CAP";ADDR$
30 X=INSTR(ADDR$,"CH")
40 CITY$=LEFT$(ADDR$,X—2)
50 PRINT "La città e' ";CITY$
60 GOTO 20
```

PAROLE COLORATE

La parola chiave COLOR verrà trattata dettagliatamente nel prossimo capitolo, ma una delle sue funzioni riveste un certo interesse nell'ambito della elaborazione di testi. Potete specificare il colore in cui il testo che segue dovrà essere stampato.

Battete COLOR seguito da uno dei numeri elencati nella Tabella 4.1 (supponendo che vi troviate nella modalità di risoluzione media e che abbiate un monitor a colori) ed il testo verrà visualizzato nel colore corrispondente. (*Nota*: questi sono i colori di default. Essi saranno diversi se avete ridefinito i colori dal Pannello di Controllo). Il seguente programma visualizzerà la parola *Colorato* in 16 colori diversi. Una delle parole non sarà visibile, in quanto il suo colore corrisponderà a quello dello sfondo, tuttavia la maggior parte delle parole sarà visibile e sarà disposta in diagonale. Lanciate il programma per vedere l'effetto.

```
 5 REM Parole colorate
10 FULLW 2: CLEARW 2
20 FOR I=0 TO 15
30 COLOR I: PRINT TAB(I*2) "Colorato"
40 NEXT
```

Il prossimo esempio di programma impiega il comando GOTOXY per visualizzare asterischi in posizioni casuali nello schermo usando i colori in modo casuale. Incontreremo ancora spesso la funzione RND nei programmi di grafica.

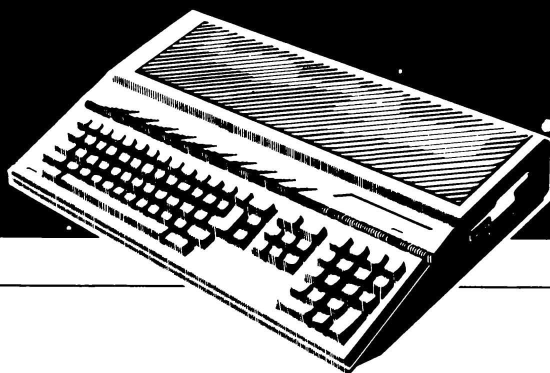
```
 5 REM Stampare asterischi in posizioni casuali
10 FULLW 2: CLEARW 2
20 GOTOXY INT (RND*34),INT(RND*17)
30 COLOR INT (RND*15)
40 PRINT "*";:GOTO 20
```

Vediamo un ultimo divertente esempio di programma. Questo riporta sullo schermo la faccia di Bob. Purtroppo questi Bob non si sentono molto bene e ne consegue che le loro facce cambiano continuamente colore (come vedrete quando lancerete il programma).

```
 5 REM Bob non sta bene
10 FULLW 2: CLEARW 2
20 FOR C=1 TO 15; R=3*INT(RND*3): GOTOXY 10+R,5
30 PRINT CHR$(28);CHR$(29)
40 GOTOXY 10+R,6 PRINT CHR$(30);CHR$(31)
50 COLOR C: NEXT
60 GOTO 20
```

SOMMARIO

Le funzioni matematiche e quelle di testo non costituiscono l'uso più entusiasmante che possiate fare di un computer potente come l'ST, ma restano ciò nondimeno molto importanti. Anche se un comando che svolge delle funzioni matematiche elementari non colpisce come un cubo o un paesaggio tridimensionale tracciati sullo schermo, non potreste procedere a fare cose più interessanti senza conoscere prima i comandi fondamentali. Se avete compreso ciò che ho trattato in questo capitolo, è arrivato il momento di passare a qualcosa di più divertente ed utile: la grafica dell'Atari ST.



GRAFICA ST

Una delle caratteristiche più notevoli dei computer ST è la grafica. Per *grafica* s'intendono le linee, i quadrati, i cerchi e tutte le altre figure che il computer può disegnare sullo schermo. I primi personal computer, quali il TRS-80 Modello 1, disponevano di una grafica in bianco e nero che ottenevano con dei punti tracciati - *pixel* - che erano piuttosto grandi e quindi qualsiasi figura disegnata sullo schermo pareva grezza e approssimativa. In breve, un cerchio non aveva mai l'aspetto di un cerchio.

Computer più recenti, come l'Atari ST, possono disporre della grafica *ad alta risoluzione*. I pixel sono abbastanza numerosi da rendere le figure piuttosto ben definite. Inoltre l'Atari può produrre fino a 512 colori diversi, rendendo la grafica del computer ancora più versatile.

Quasi tutte le parole chiave del BASIC Atari si basano sul sistema di coordinate X,Y. Anche se i vostri ricordi di geometria si sono un po' annebbiati, creare la grafica con l'ST è lo stesso molto facile con tale metodo. Potete indirizzare qualsiasi punto dello schermo semplicemente usando due numeri, la coordinata X (che dice dove si trova il pixel in senso orizzontale) e la coordinata Y (che ne indica la posizione in verticale). Un pixel che si trova sul bordo sinistro dello schermo ha l'ascissa X pari a 0, dato che l'asse delle X inizia sulla sinistra e finisce a 303, 607 o 615 a destra (parleremo tra breve della differenza tra questi valori). L'asse Y, invece, inizia a 0 nella parte alta dello schermo e finisce a 166 o 344 nella parte inferiore. Usando le coordinate (X,Y) potete individuare qualsiasi pixel, come si può vedere nella Figura 4.1.

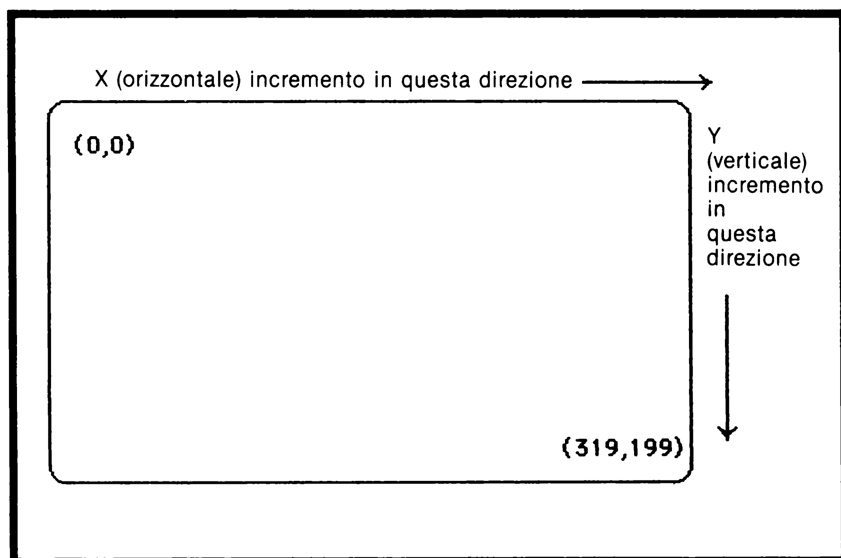


Figura 4.1: uso delle coordinate X e Y per localizzare un pixel.

I pixel possono essere *accesi* o *spenti*. Avendo molti pixel *accesi* contemporaneamente, si possono ottenere linee, quadrati, cerchi, forme tridimensionali, figure e grafici. Se volete accendere il pixel nell'angolo in alto a sinistra dello schermo basta specificare il pixel (0,0) nel comando grafico. Nella modalità a bassa risoluzione, il pixel nell'angolo in basso a destra ha coordinate (303,166). Il centro dello schermo ha invece coordinate (151,83).

Non esiste un'istruzione del BASIC ST che accenda o spenga direttamente un certo pixel. Farete comunque uso delle coordinate per stabilire in quale punto dello schermo andrà collocata una certa forma. Per esempio, per disegnare una linea, si dice al computer di disegnare una linea da un certo punto ad un altro, come nella Figura 4.2. Nel caso di un cerchio, specificherete la posizione del centro e la lunghezza del raggio (misurato in pixel).

RISOLUZIONI

Come abbiamo già accennato nel Capitolo 1, l'opzione Set Preferences della barra dei menù della scrivania GEM vi permette di specificare se volete porvi in alta risoluzione (per monitor monocromatici) o in risoluzione media o bassa (per monitor a colori). Se impiegate un monitor a colori, potete scegliere liberamente tra la risoluzione media e quella bassa, quando scrivete i vostri pro-

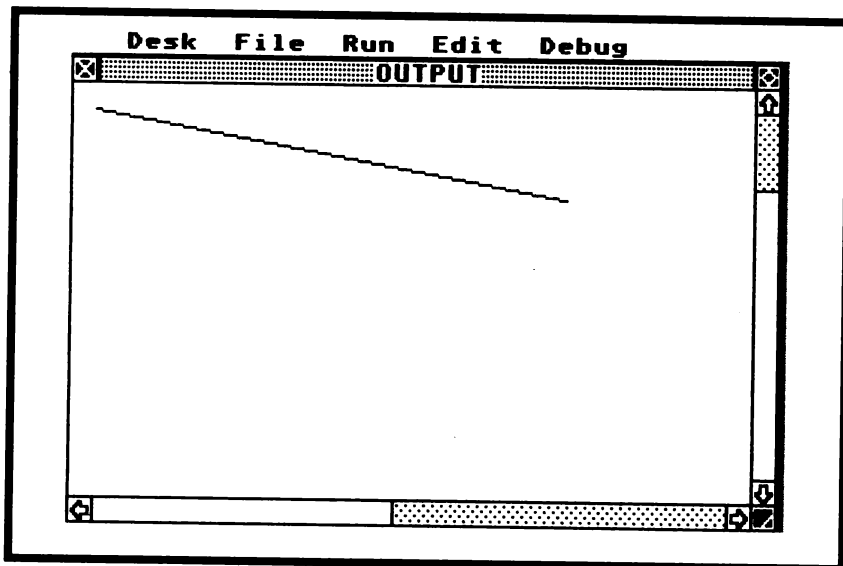


Figura 4.2: una linea.

grammi di grafica. La risoluzione media ha il doppio di pixel nella coordinata X (in altre parole potete accendere pixel di ascissa pari fino a 607) e potete usare solo un massimo di quattro colori contemporaneamente. Con la risoluzione bassa l'immagine non sarà altrettanto dettagliata, ma potrete scegliere tra 16 colori diversi. Con il monitor monocromatico ad alta risoluzone, naturalmente, avrete una grafica molto dettagliata (risoluzione di 615 per 344), ma solo un "colore": bianco e nero.

In questo libro, userò quasi esclusivamente la risoluzione bassa (usate Set Preferences per passare alla bassa risoluzione, in modo da poter seguire gli esempi). La qualità della grafica a bassa risoluzione rimane soddisfacente, inoltre la possibilità di usare 16 colori rende la grafica molto più interessante (anche se tutte le illustrazioni di questo libro sono stampate in bianco e nero).

Un'ultima nota, prima di passare a trattare nei dettagli la grafica dell'ST. Alcuni dei programmi vi parranno a prima vista un po' scemi, ma lo scopo di ciascuno di essi è d'insegnarvi qualche aspetto dei comandi grafici del BASIC ST. Alcuni tracceranno sullo schermo disegni interessanti, creeranno colori a caso, o tracceranno linee da un punto ad un altro, ma ciascuno di essi contribuirà alla vostra conoscenza del BASIC ST; e poiché state imparando per esperienza, diventerete presto abbastanza esperti da poter scrivere i vostri (forse più sofisticati) programmi di grafica.

TRACCIARE LINEE

Su molti personal computer, l'istruzione grafica più elementare disegna un punto sullo schermo. Nel BASIC ST l'istruzione più semplice è LINEF, che traccia una linea retta tra un punto ed un altro. Usando diverse istruzioni LINEF, potete disegnare oggetti più complessi, come quadrati e cubi (per inciso, se volete disegnare un punto, basta usare LINEF per tracciare una "linea" da un certo punto al punto stesso).

Il formato di LINEF è:

LINEF *X1,Y2,X2,Y2*

Qui non ci sono le parentesi né altri simboli speciali da ricordare; basta battere *LINEF* seguito dalle coordinate in pixel del punto in cui volete iniziare la linea *(X1,Y1)* e le coordinate in pixel del punto in cui deve finire *(X2,Y2)*. Per esempio, per tracciare una linea da (20,30) a (60,43) si batta *LINEF 20,30,60,43*.

Il primo programma traccia a caso una linea interrotta che va ad occupare tutto lo schermo. Ogni segmento che la compone inizia dove il precedente si interrompe e l'Atari ricorda le ultime coordinate impiegate memorizzandole

rispettivamente nelle variabili OX e OY. Anche il colore di ciascun segmento è casuale e lo schermo sarà presto pieno di decine di segmenti colorati. Quando vorrete interrompere il programma, premete Control-C.

```
 5 REM Disegnare una linea ininterrotta casuale
10 FULLW 2: CLEARW 2
20 X=RND*300: Y=RND*150
30 LINEF OX,OY,X,Y
40 OX=X: OX=Y
50 COLOR 1,1,RND*15
60 GOTO 20
```

Quando si usa LINEF, di solito si vuole avere più controllo su dove collocare le linee. Le istruzioni READ...DATA, i loop FOR...NEXT e le istruzioni INPUT permettono di specificare dove le linee andranno collocate per tracciare un disegno preciso. Il prossimo programma, per esempio, impiega due loop FOR...NEXT per creare un interessante disegno con valori variabili di X e Y:

```
 5 REM Usa due loop FOR...NEXT per tracciare diagonali
10 FULLW 2: CLEARW 2
20 FOR X=0 TO 300 STEP 20
30 FOR Y=0 TO 150 STEP 10
40 LINEF O,O,X,Y
50 NEXT: COLOR 1,1,X/20: NEXT
```

Per tracciare righe verticali ed orizzontali (anziché diagonali), i loop FOR...NEXT vanno separati, anziché annidati.

Supponiamo che vogliate creare un reticolo per tracciare un piccolo diagramma lineare. Dovrete prima tracciare diverse linee orizzontali, distanti uniformemente e opportunamente allineate rispetto al bordo. Traccerete poi lo stesso numero di linee verticali, in modo che con le linee orizzontali formino i quadretti del reticolo. Il loop FOR...NEXT è perfetto a questo scopo, perché non si limita a contare le linee, ma le può anche collocare impiegando la sua variabile contatore come ascissa (o ordinata) X (o Y) nell'istruzione LINEF. Per cominciare cancellate lo schermo e stabilite il primo loop per tracciare le linee verticali:

```
 5 REM Uso di FOR...NEXT per creare un reticolo
10 FULLW 2: CLEARW 2: COLOR 1,1,1
20 FOR X=0 TO 200 STEP 10
30 LINEF X,0,X,150: NEXT
```

La variabile X nell'istruzione LINEF cambia a seguito del loop FOR...NEXT e le due coordinate di Y (0 e 150) rimangono costanti. Non c'è alcuna necessità che il valore di Y cambi, perché già avete un ben definito limite superiore ed inferiore per ciascuna linea verticale. Il valore di X, invece, deve variare per spaziare le linee verticali di 10 pixel sull'asse delle X. Ora che le linee verticali sono tracciate, il resto del programma può tracciare quelle orizzontali:

```
40 FOR Y=0 TO 150 STEP 10
50 LINEF 0,Y,200,Y: NEXT
60 GOTO 60
```

L'istruzione GOTO 60 della linea 60 "fissa" il grafico in modo che abbiate l'opportunità di guardarlo senza che debba ricomparire sullo schermo la finestra Command; se volete uscire da questo loop infinito, premete Control-C.

Potete allungare di un passaggio questa applicazione e creare un diagramma lineare casuale sulla "carta millimetrata" che abbiamo già tracciato. La prima parte di questo programma è uguale a quella dell'esempio precedente:

```
 5 REM Creare una diagramma lineare casuale
10 FULLW 2: CLEARW 2: COLOR 1,1,1
20 FOR X=0 TO 200 STEP 10
30 LINEF X,0,X,150: NEXT
40 FOR Y=0 TO 150 STEP 10
50 LINEF 0,Y,200,Y: NEXT
```

Successivamente disegneremo una linea entro il perimetro dell'area del reticolo. Poiché il diagramma si sposterà nel senso dell'asse delle X e avrà valori di Y (verticali) casuali, il loop FOR...NEXT cambierà i valori della X. In questo caso il loop FOR...NEXT si sposta con incrementi di 5, e una volta che venga scelto un valore casuale di Y, il computer traccia un segmento dalle vecchie coordinate alla nuova posizione avente ascissa di 5 pixel più a destra rispetto al vecchio punto e ordinata casuale. Anche qui il computer deve memorizzare il vecchio valore (in questo caso il valore della Y) in un'altra variabile in modo che possa sapere dove cominciare a disegnare il segmento successivo.

```
 60 COLOR 1,1,5
 70 FOR X=0 TO 195 STEP 5
 80 Y1=RND*150: LINEF X,Y2,X+5,Y1
 90 Y2=Y1: NEXT
100 GOTO 100
```

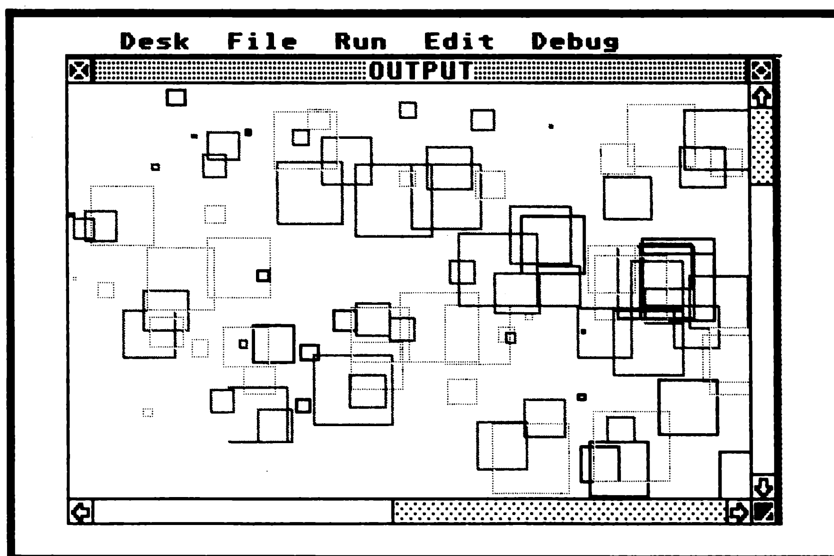


Figura 4.3: quadrati casuali.

Creare un quadrato o un rettangolo con le istruzioni LINEF è altrettanto facile che tracciare linee. I lati superiore, inferiore, destro e sinistro del quadrilatero dovranno essere coerenti, nel senso che dovranno avere in comune quattro punti che corrispondono ai vertici. Se volete che un quadrilatero abbia l'angolo in alto a sinista in (X1,Y1), e l'angolo in basso a destra in (X2,Y2), battete quanto segue:

```
LINEF X1,Y1,X2,Y1: LINEF X2,Y1,X2,Y2: LINEF X2,Y2,X1,Y2: LINEF X1,Y2,X1,Y1
```

In ciascuna istruzione LINEF il secondo punto coincide con il primo dell'istruzione LINEF successiva. Nel programma illustrato dalla Figura 4.3, vengono selezionate ascisse e ordinate casuali e vengono poi memorizzate nelle variabili X e Y. Successivamente, vengono memorizzate una larghezza (variabile W) e una lunghezza casuali (variabile L, che corrisponde a 1.2 volte la larghezza W, poiché un pixel non è perfettamente quadrato). Con questa informazione il computer può tracciare quadrati di disposizione, larghezza, lunghezza e colore casuali.

```
 5 REM Disegnare quadrati a caso
10 FULLW 2: CLEARW 2
20 X=RND*300: Y=RND*150
```

```
30 L=RND*30: W=L*1.2
40 LINEF X,Y,X+W,Y
50 LINEF X+W,Y,X+W,Y+L
60 LINEF X+W,Y+L,X,Y+L
70 LINEF X,Y+L,X,Y
80 COLOR 1,1,INT(RND*15)+1: GOTO 20
```

Anche se molti tipi diversi di quadrati riempiono lo schermo, responsabili di ciascuno di essi sono solo le quattro istruzioni LINEF , dato che tutte le variabili sono fluttuanti.

Invece di avere quadrati completamente casuali, potreste creare quadrati geometricamente simili, ma di dimensioni diverse. Il prossimo programma impiega un loop FOR...NEXT per tracciare 40 quadrati di dimensioni crescenti. I primi quadrati sono molto piccoli, ma man mano che il loop FOR...NEXT procede, le dimensioni di ogni quadrato sono maggiori rispetto a quelle del precedente.

Ciò crea un interessante effetto tridimensionale, come possiamo vedere nella Figura 4.4.

Quando il disegno è finito, l'ST lo traccia di nuovo in un colore diverso; lanciate il programma per vedere i colori. Il programma continuerà a dare al disegno un nuovo colore finché non fermerete il programma con Control-C.

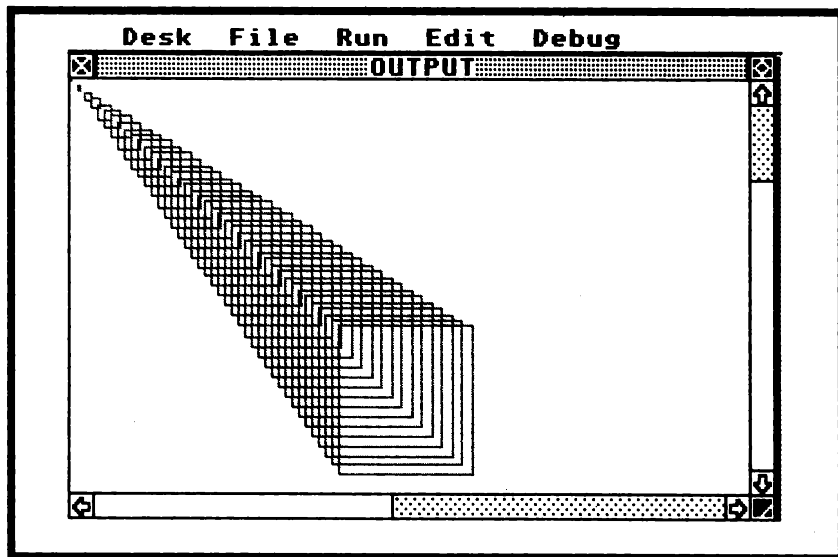


Figura 4.4: quadrati in sequenza.

```
 5 REM Disegnare quadrati ordinatamente crescenti
10 FULLW 2: CLEARW 2
20 FOR X=0 TO 120 STEP 3
30 Y=X/1.2
40 LINEF X,Y,X+X/2,Y
50 LINEF X+X/2,Y,X+X/2,Y+X/2
60 LINEF X+X/2,Y+X/2,X,Y+X/2
70 LINEF X,Y+X/2,X,Y
80 NEXT
90 COLOR 1,1,INT(RND*15)+1: GOTO 20
```

Come probabilmente avrete ormai capito, con l'istruzione LINEF potrete creare qualsiasi disegno composto di segmenti lineari, purché possiate specificare in quale punto preciso esso va collocato. A volte dovrete conoscere le posizioni *assolute* (quando tracciate un grafico ad istogrammi, è molto importante che la collocazione sia accurata); altre volte invece basterà conoscere la *posizione relativa* dei punti per poter disegnare una forma di qualsiasi dimensione in qualsiasi punto dello schermo.

Potete tracciare un triangolo, per esempio, sapendo che i primi due vertici devono trovarsi sulla stessa linea ed il terzo dovebbe trovarsi tra i primi due, ma ad una certa altezza; in base a questo genere di rozza formula, potreste scrivere un programma di sole poche righe che disegni centinaia di triangoli sullo schermo.

Il prossimo programma impiega le coordinate assolute per tracciare sullo schermo una piramide a base triangolare tridimensionale. Ogni volta che lancerete il programma, comparirà la stessa forma. Potete provare a scrivere un programma che possa tracciare la stessa piramide per tutto lo schermo, ma di dimensioni diverse e collocata in modo casuale.

```
 5 REM Disegnare una piramide
10 FULLW 2: CLEARW 2
20 LINEF 50,90,100,90
30 LINEF 75,20,50,90
40 LINEF 75,20,100,90
50 LINEF 50,90,75,80
60 LINEF 75,80,100,90
70 LINEF 75,80,75,20
```

Una delle funzioni pratiche più ovvie dell'istruzione LINEF è quella di creare grafici più complessi. Il prossimo programma servirà a creare il diagramma ad istogrammi della Figura 4.5. Invece di tracciare ancora il reticolo, traccia-

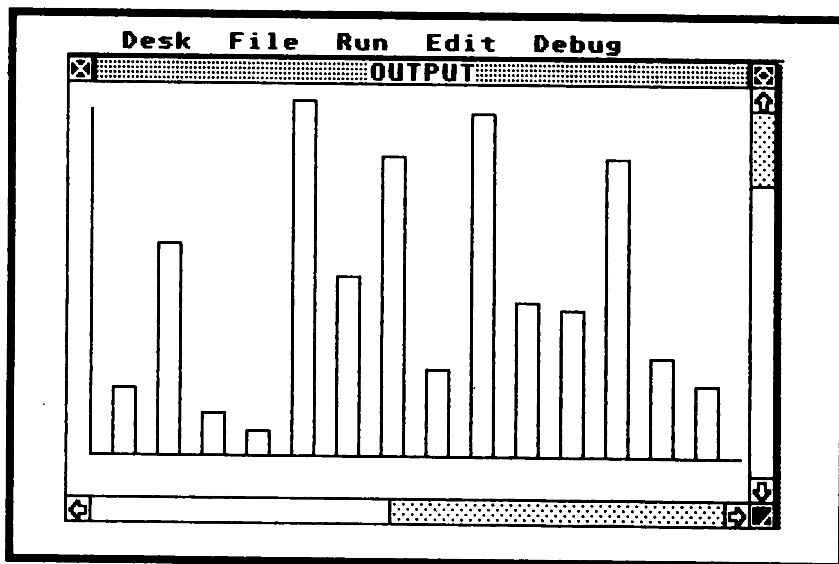


Figura 4.5: un grafico ad istogrammi.

mo gli assi delle X e delle Y per avere un'idea di come confrontare gli istogrammi tra loro:

```
 5 REM Disegnare un diagramma a istogrammi
10 FULLW 2: CLEARW 2
20 LINEF 10,10,10,150
30 LINEF 10,150,300,150
```

Poi il valore di X passa da 20 a 280 con incrementi di 20; ciò significa che ogni istogramma del grafico sarà spostato di 20 pixel. Per questo motivo, gli istogrammi dovranno essere larghi meno di 20 pixel, per fare in modo che non si sovrappongano. Una volta che venga selezionato un valore casuale di Y (l'altezza dell'istogramma), l'ST traccerà: (1) una linea dall'asse X all'altezza determinata dal valore della variabile Y, (2) una linea da questo valore di Y verso destra di 10 pixel (da X a X+10) e (3) una linea che ritorna da questo all'asse X. Questo procedimento completa un istogramma e ciascuno di essi sarà largo 10 pixel. Una volta che il loop ha finito (ovvero quando X raggiunge il valore 280), il diagramma ad istogrammi è completo.

```
40 FOR X=20 TO 280 STEP 20
50 Y=INT(RND*145)
60 LINEF X,150,X,Y
```

```
 70 LINEF X,Y,X + 10,Y
 80 LINEF X + 10,Y,X + 10,150
 90 NEXT
100 GOTO 100
```

Come con qualsiasi parola chiave, il modo migliore d'imparare meglio LINEF è quello di usarla. Cercate di tracciare poligoni diversi, figure tridimensionali e grafici impiegando solo l'istruzione LINEF. Anche se svolge un solo compito, LINEF si può impiegare utilmente per molti tipi di grafici.

Nel Capitolo 8, userò LINEF in un programma più realistico, ma anche più complicato, per tracciare diagrammi lineari o a istogrammi basati su valori dati, anziché casuali.

COLORI, COLORAZIONE DEL FONDO E FORME

L'istruzione COLOR

L'istruzione COLOR, presentata nel Capitolo 3, vi permetterà di cambiare il colore del testo, ma non solo. Il formato di COLOR è:

COLOR *[colore del testo,colorazione del fondo,colore della linea,stile,indice]*

I primi tre argomenti sono numeri che rappresentano un colore da usare con i vari comandi grafici. Gli ultimi due argomenti rappresentano un certo stile di riempimento degli oggetti disegnati.

Notate che tutti questi argomenti vengono presentati come opzionali. In realtà, almeno uno è necessario in una istruzione COLOR, secondo il contesto. Segue una descrizione più dettagliata.

Argomenti di COLOR

Colore del testo Come sapete, l'argomento colore del testo cambia il colore del testo stampato.

I numeri usati per specificare il colore possono variare da 0 a 15 (se vi trovate nella modalità a bassa risoluzione). Se scegliete la risoluzione media, il numero del colore può variare da 0 a 3. Per i monitor monocromatici ad alta risoluzione, potrete usare solo 0 o 1. (*Nota Bene:* ad ogni colore non è assegnato un numero fisso. Potete assegnare un colore ad un certo numero usando il Pannello di Controllo.)

Colorazione del fondo. Con l'istruzione FILL, di cui parleremo tra breve, potete riempire un'area grafica con un colore. Il colore che scegliete è specificato nell'istruzione COLOR da questo secondo argomento.

Colore della linea. L'argomento colore della linea dice al computer di che colore dovranno essere le linee. Le linee non sono solo quelle tracciate con l'istruzione LINEF; possono anche essere le linee che costituiscono un cerchio, un'ellisse o altre figure.

Stile,Indice. Questi ultimi due numeri, in effetti, costituiscono un unico argomento. L'ST dispone di un insieme di *colorazioni del fondo* in memoria, individuati da una coppia unica di numeri *stile,indice.* Le coppie di numeri vanno da (1,2) a (24,2) e da (1,3) a (12,3) e sono illustrate nella Figura 4.6.

Come cambiare i colori

Non dimenticate che quando date un'istruzione COLOR, le immagini sullo schermo in quel momento non cambieranno; COLOR può specificare il colore solo della grafica e del testo *che seguono.* I numeri impiegati con la parola chiave COLOR sono dati nella Tabella 4.1.

L'istruzione COLOR non si limita a rendere la grafica più interessante e più

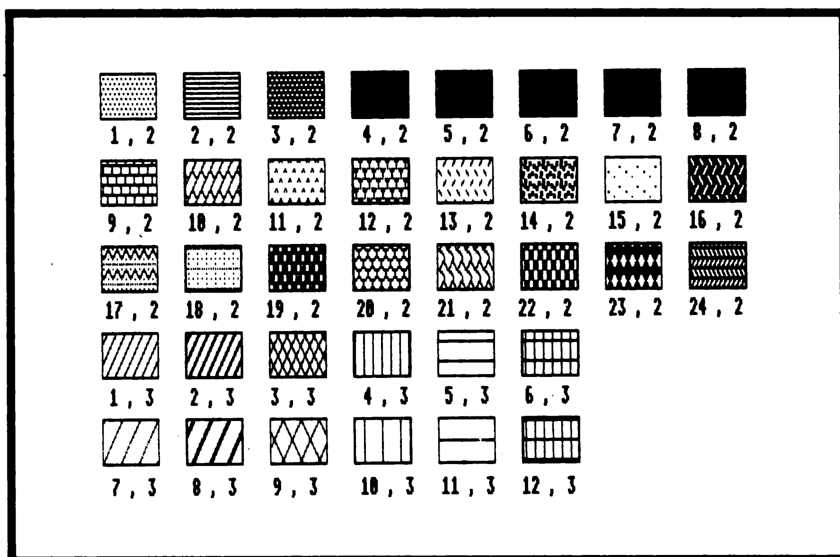


Figura 4.6: colorazioni del fondo e relative coppie di numeri.

divertente; la rende anche più utile. Per esempio, se volete disegnare un diagramma circolare che rappresenti diverse industrie, potete separare i diversi settori del diagramma usando diversi colori o diverse colorazioni del fondo o entrambi. Più avanti in questo capitolo imparerete anche ad adattare i 16 colori che la tavolozza vi offre.

L'istruzione FILL

Una parola chiave grafica che opera in stretta connessione con COLOR è FILL, il cui formato è:

```
FILL X,Y
```

In questa istruzione, X,Y specificano il punto nel quale iniziare a riempire l'area con un colore (come specificato nell'istruzione COLOR). Poiché viene riempita l'intera area, qualsiasi punto al suo interno può servire come *X,Y.* Tuttavia, FILL riempirà un'area con il colore *solo* se quell'area è completamente delimitata da pixel. FILL funziona come una specie di secchio di vernice computerizzato che continuerà a riempire l'area delimitata fino a raggiungerne i bordi. Tuttavia, se manca anche un solo pixel dal bordo, permettendo

Numero	Colore	Numero	Colore
0	bianco	8	grigio
1	nero	9	grigio scuro
2	rosso	10	ottanio (blu verde)
3	verde	11	ottanio scuro
4	azzurro	12	magenta (porpora chiaro)
5	blu	13	magenta scuro
6	rosso scuro (marrone)	14	giallo
7	verde scuro	15	giallo scuro

Tabella 4.1: i numeri di default dei colori.

al colore di “fuoriuscire”, esso continuerà a diffondersi fino a incontrare un nuovo confine.

Il prossimo programma è identico al programma grafico precedente, salvo che aggiunge le istruzioni COLOR e FILL e per delineare meglio gli istogrammi, li colora di verde. Trovare il punto giusto per iniziare il riempimento è stato facile, poiché il computer usa il punto che si trova spostato di un pixel verso destra e di un pixel verso l’alto rispetto al suo corrente punto d’inizio per ciascun istogramma; quindi il punto si trova certamente entro il contorno dell’istogramma.

```
  5 REM Disegnare un diagramma a istogrammi casuale
 10 FULLW 2: CLEARW 2: COLOR 1,3,1,1,1
 20 LINEF 10,10,10,150
 30 LINEF 10,150,300,150
 40 FOR X=20 TO 280 STEP 20
 50 Y=INT(RND*145)
 60 LINEF X,150,X,Y
 70 LINEF X,Y,X+10,Y
 80 LINEF X+10,X,X+10,150
 90 FILL X+1,149: NEXT
100 GOTO 100
```

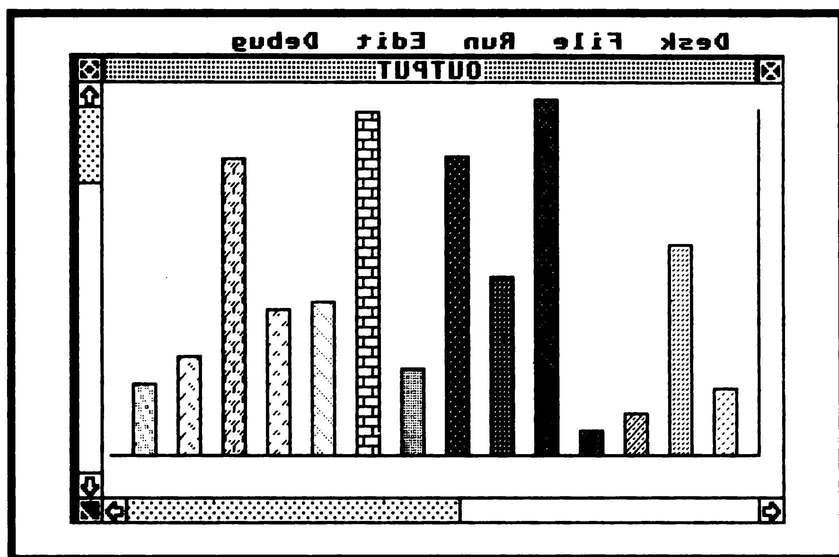


Figura 4.7: grafico ad istogrammi con diversi colori e colorazioni del fondo.

Come ho accennato prima, potete differenziare ulteriormente le diverse parti di un grafico sia usando colori diversi, sia usando colorazioni del fondo. Il prossimo programma per diagramma ad istogrammi sceglie colori e fondo diversi per ciascun istogramma. I risultati sono illustrati nella Figura 4.7.

```
  5 REM Usare colori e colorazioni del fondo diversi in un diagramma
    a istogrammi
 10 C=1: FULLW 2: CLEARW 2: COLOR 1,2,1,1,2
 20 LINEF 10,10,10,150
 30 LINEF 10,150,300,150
 40 FOR X=20 TO 280 STEP 20
 50 Y=INT(RND*145)
 60 LINEF X,150,X,Y
 70 LINEF X,Y,X+10,Y
 80 LINEF X+10,Y,X+10,150
 90 FILL X+1,149: C=C+1: COLOR 1,C,1,C,2: NEXT
100 GOTO 100
```

La variabile C assicura che ciascun motivo e colore sia unico per ciascun istogramma nel grafico. Ogni volta che viene tracciato un istogramma, la variabile aumenta (il suo valore viene incrementato di 1) in modo che l'istogramma successivo avrà il successivo colore e motivo,come specificato nell'istruzione COLOR.

CERCHI ED ELLISSI

Un altro vantaggio delle parole chiave grafiche del BASIC ST è che vi permettono di disegnare forme complesse con facilità. Con alcuni computer si devono svolgere calcoli complicati per disegnare un semplice cerchio. Il BASIC ST, invece, ha istruzioni come CIRCLE, PCIRCLE e ELLIPSE, con le quali tracciare un cerchio o un'ellisse è facile come disegnare una linea.

L'istruzione CIRCLE

L'istruzione CIRCLE disegna una circonferenza perfetta, (o un arco di circonferenza). Il suo formato è:

CIRCLE *X,Y,raggio[,angolo iniziale,angolo finale]*

I primi tre elementi X,Y e *raggio* sono necessari per usare CIRCLE. X e Y sono l'ascissa e l'ordinata del centro del cerchio. Come sempre, queste coordinate sono misurate dai pixel dello schermo, quindi un cerchio posto al centro dello schermo avrà coordinate del centro di circa (151,83) nella bassa risoluzione. Il raggio, anch'esso misurato in pixel, stabilisce la distanza tra il centro del cerchio e la sua circonferenza. *CIRCLE 60,73,20* disegnerà un cerchio di 20 pixel di raggio, centrato nel punto (60,73).

Gli argomenti *angolo iniziale* e *angolo finale* sono opzionali. Se decidete di usarli, essi dicono al computer di tracciare un *arco* (una parte della circonferenza di un cerchio) iniziando in un certo punto del cerchio per finire in un altro punto preciso. Gli angoli nel cerchio vanno da 0 (il bordo più a destra a 1800 e quello più in basso a 2700. Ricordate che l'angolo di un cerchio misura 360 gradi (che, moltiplicato per 10 dà 3600, il numero totale dei punti stra a 1800 e quello più in basso a 2700. Ricordate che l'angolo di un cerchio misura 360 gradi (che, moltiplicato per 10 da' 3600, il numero totale dei punti che potete specificare nel cerchio) quindi potete specificare angoli con una precisione di 0.1 gradi. Se volete disegnare un cerchio con il centro in (20,30), raggio di 55, un angolo iniziale di 260 gradi e un angolo finale di 322 gradi, basta battere questo:

```
CIRCLE 20,30,55,2600,3220
```

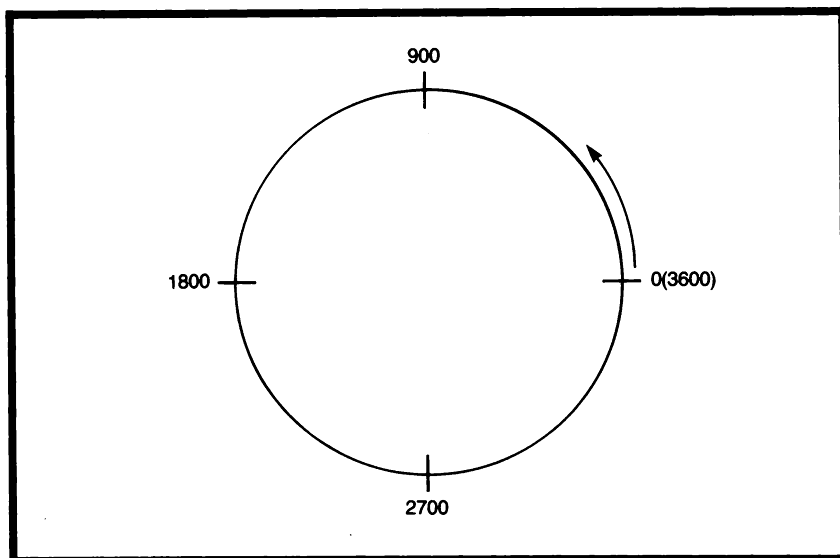


Figura 4.8: posizione dei valori degli angoli in un cerchio.

È importante ricordare di moltiplicare per 10 la misura dell'angolo.

I programmi seguenti illustrano il significato degli argomenti di CIRCLE, dimostrando gli effetti che si producono al variare del valore di uno o di più argomenti. Vi mostreranno anche come creare alcuni grafici molto interessanti.

Il primo programma CIRCLE varia il raggio per tracciare una serie di cerchi concentrici (hanno tutti lo stesso centro) che prima si allargano e poi si restringono, come nella Figura 4.9. Dato che il colore di ciascun cerchio è casuale, potete seguire agilmente l'allargarsi e il restringersi dei cerchi. Anche se viene impiegato un solo loop FOR...NEXT, il programma può far andare i cerchi in entrambe le direzioni con l'aiuto della variabile D. La variabile D passa da 0 a 1 e poi di nuovo a 0 nella linea 60. Quando D è uguale a 0 i cerchi vanno verso l'esterno, poiché la variabile D fa parte del raggio specificato nella linea 30; quando D è uguale a 1, il cerchio diventa sempre più piccolo.

```
 5 REM Disegnare cerchi concentrici
10 FULLW 2: CLEARW 2
20 FOR I=3 TO 75 STEP 3
30 CIRCLE 150,78,D*78—I
40 COLOR 1,1,INT(RND*15)+1
50 NEXT
60 IF D=0 THEN D=1 ELSE D=0
70 GOTO 20
```

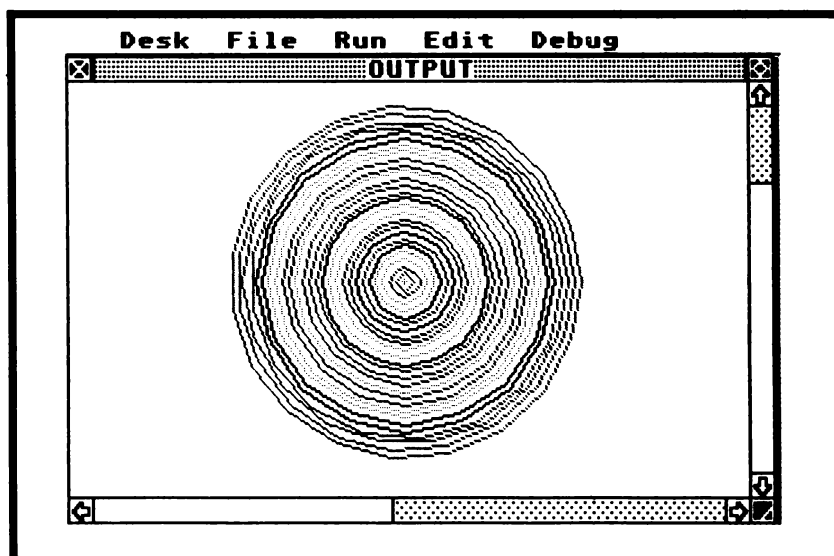


Figura 4.9: cerchi concentrici.

L'uso di un alternatore come quello della linea 60, può essere una tecnica molto potente nella programmazione, perché vi evita di raddoppiare le linee di programma per ottenere due diversi effetti. Ogni volta che incontra la linea, il computer controlla il valore della variabile; se è uguale ad un valore, lo trasforma nell'altro, e viceversa. Questo scambio del valore di una variabile può poi essere usato in altre parti del programma, come nel loop FOR...NEXT dell'esempio precedente.

Il prossimo esempio è un programmino simpatico che dimostra come posizionare i cerchi, variando i punti X,Y (il centro). I loop annidati FOR...NEXT di X e Y creano cerchi che riempiono l'intero schermo ed i centri dei cerchi sono allineati in modo da creare contemporaneamente un motivo regolare interessante, come nella Figura 4.10.

```
 5 REM Posizionare cerchi
10 FULLW 2: CLEARW 2
20 FOR X=0 TO 300 STEP 15
30 FOR Y=0 TO 165 STEP 15
40 CIRCLE X,Y,14
50 NEXT: NEXT
60 GOTO 60
```

Gli effetti speciali che si possono ottenere intersecando dei cerchi sono illu-

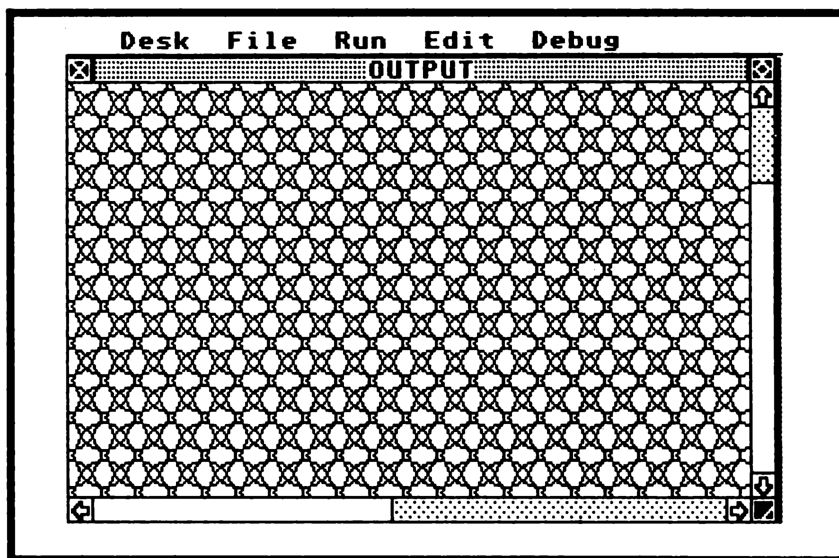


Figura 4.10: combinazione di cerchi.

strati dal prossimo programma, che varia sia il centro che il raggio.Cinque cerchi di dimensioni medie formano un fiore, basta posizionarli opportunamente impiegando uno o due loop. Il primo loop fa allargare ciascun cerchio verso l'esterno, mentre il loop più interno sposta i cinque disegni per lo schermo

```
  5 REM Cerchi sovrapposti
 10 FULLW 2: CLEARW 2
 20 REM Inizio del loop annidato
 30 FOR I=0 TO 70 STEP 3
 40 FOR J=—20 TO 20 STEP 20
 50 CIRCLE 148-(J*2),80—J,I
 60 CIRCLE 148+J*2,80—J,I
 70 NEXT J
 80 COLOR 1,1,INT(RND*15)+1
 90 NEXT
100 GOTO 30
```

Proprio come avete creato effetti tridimensionali con l'istruzione LINEF tracciando quadrati sempre più grandi, potete disegnare oggetti quali i coni usando l'istruzione CIRCLE; ogni cerchio tracciato deve avere il raggio leggermente più grande ed il centro spostato in basso a destra rispetto a quello

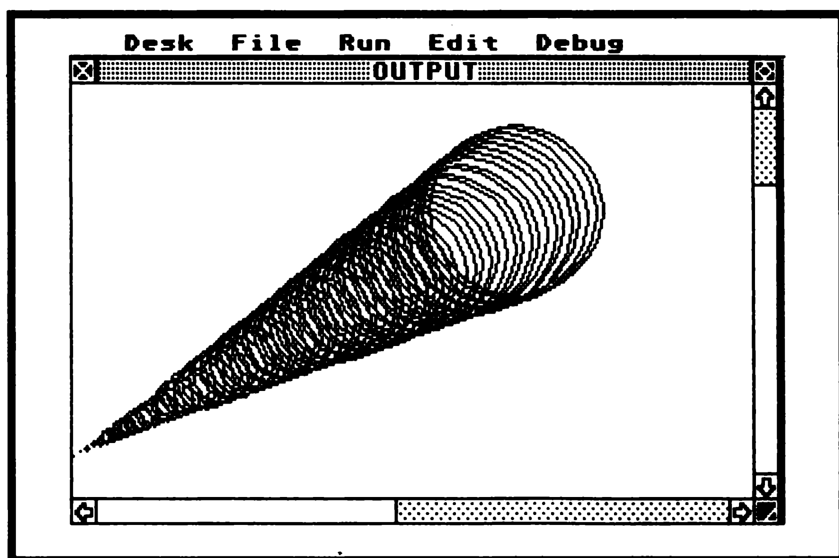


Figura 4.11: un cono.

precedente. Una volta che ci siano un certo numero di cerchi sullo schermo,l'effetto della profondità del cono sarà evidente, come nella Figura 4.11.

```
 5 REM Disegnare un cono
10 FULLW 2: CLEARW 2
20 FOR X=0 TO 200 STEP 3
30 CIRCLE X,150—X/2,X/5
40 NEXT
50 GOTO 50
```

Vediamo una variazione della stessa idea, con due coni multicolori che partono dai due angoli inferiori dello schermo e si incrociano verso il centro in un cerchio appartenente ad entrambi:

```
 5 REM Coni incrociati
10 FULLW 2: CLEARW 2
20 FOR X=0 TO 150 STEP 3
30 CIRCLE X,150—X/2,X/5
40 CIRCLE 300—X,150—X/2,X/5
50 COLOR 1,1,INT(RND*15)
60 NEXT
70 GOTO 70
```

L'angolo iniziale e finale opzionali nell'istruzione CIRCLE si usano per creare diagrammi circolari, disegni e archi. Nel prossimo programma creiamo due insiemi di archi concentrici colorati, con gli archi sempre più lontani dal centro man mano che il loop procede:

```
 5 REM Archi concentrici
10 10 FULLW 2: CLEARW 2
20 FOR I=0 TO 70 STEP 3
30 X=INT(RND(1)*3400): CIRCLE 70,80,I,X,X+300
40 CIRCLE 240,80,I,X,X+300
50 COLOR 1,1,INT(RND(1)*15)+1
60 NEXT
70 GOTO 20
```

Potete anche conferire un effetto di "movimento" agli archi allontanandoli in modo uniforme rispetto al centro e accendendoli e spegnendoli in sequenza. Vediamo ora un esempio che prima traccia un cerchio completo, poi traccia archi di 15 gradi, in senso antiorario lungo la circonferenza. Poiché il co-

lore degli archi è quasi sempre diverso da quello del cerchio originario, potete seguire gli archi che girano sulla circonferenza:

```
 5 REM Archi ruotanti
10 FULLW 2: CLEARW 2
20 CIRCLE 150,80,70
30 FOR A=0 TO 3500 STEP 100
40 CIRCLE 150,80,70,A,A+150
50 COLOR 1,INT(RND*15),INT(RND*15)
60 NEXT
70 GOTO 30
```

Un altro effetto speciale è la spirale, che potete creare alterando gli argomenti raggio, angolo iniziale e finale, in modo che gli archi tracciati si allontanino dal centro del cerchio e ciascuno di essi sia leggermente ruotato rispetto al precedente.

Notate come la variabile A nel loop FOR...NEXT determina raggio, angolo iniziale e angolo finale di ciascun arco. Questo illustra ancora una volta come si possa efficientemente usare una variabile per creare un motivo regolare, come mostra la Figura 4.12.

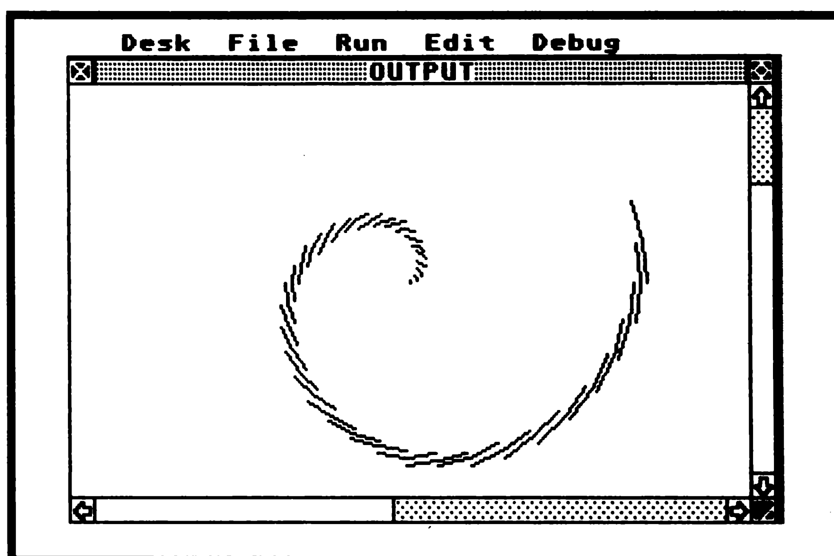


Figura 4.12: uso di una variabile per creare un motivo regolare.

```
 5 REM Uso di una variabile per creare un motivo regolare
10 FULLW 2: CLEARW 2
20 FOR A=1 TO 36
30 CIRCLE 150,80,A*3,A*100,(A+2)*100
40 NEXT
50 GOTO 50
```

Se volete che la spirale venga tracciata in modo continuo, con colori sempre diversi, inserite la seguente linea 50 in sostituzione della precedente:

```
50 COLOR 1,1,INT(RND*15): GOTO 20
```

L'istruzione PCIRCLE

PCIRCLE funziona come CIRCLE, salvo che il cerchio che tracciate con PCIRCLE è completamente colorato con il colore specificato nell'istruzione COLOR dall'argomento *colorazione del fondo.* Inoltre se disegnate solo una parte di un PCIRCLE (specificando angolo iniziale e finale) verrà tracciato non solo un arco, ma il corrispondente settore circolare. Il formato di PCIRCLE è:

PCIRCLE *X,Y,raggio[,angolo iniziale, angolo finale]*

Il prossimo programma impiega i settori circolari per creare un modello di spirale ancora più interessante rispetto a quelli che abbiamo già visto. Ad ogni giro della spirale cambia il colore.

```
 5 REM Spirale multicolore
10 FULLW 2: CLEARW 2
20 FOR A=1 TO 36
30 PCIRCLE 150,80,A*3,A*100,(A+1.5)*100
40 NEXT
50 COLOR 1,INT(RND*15): GOTO 20
```

PCIRCLE è particolarmente utile nei diagrammi circolari, dato che permette di disegnare i settori circolari. Per tracciare un diagramma circolare, mantenete costante il centro di ciascun comando PCIRCLE ed usate angoli iniziali e finali corrispondenti ai dati che volete rappresentare, vale a dire che se l'ammontare rappresentato da un settore è il 10 per cento del totale, l'angolo iniziale e l'angolo finale dovranno avere una differenza proporzionale, pari

a 360 (36 gradi). Il prossimo programma traccia un diagramma circolare in base a delle istruzioni DATA. La Figura 4.13 mostra il diagramma, ma non i colori. La prima parte del programma imposta la finestra Output e legge i cinque nomi delle variabili stringa, le colorazioni, gli angoli iniziali e finali del grafico:

```
 5 REM Diagramma circolare usando le istruzioni DATA
10 FULLW 2: CLEARW 2
20 FOR D=1 TO 5
30 READ NAME$,COL,ANI,ANF
```

Il programma poi posiziona il cursore, basandosi sul corrente valore di D. Man mano la variabile D cresce, il cursore si posiziona una linea più in basso (ma resta sempre sul bordo sinistro dello schermo). Vengono stampati i nomi delle applicazioni del computer e con COLOR vengono stabiliti i colori del testo e del fondo. Infine, i settori del diagramma vengono tracciati con PCIRCLE.

```
40 GOTOXY 0,D: COLOR COL,COL: PRINT NAME$
50 PCIRCLE 150,80,50,ANI,ANF
60 NEXT
70 GOTO 70
80 DATA Elaborazione testi,2,0,410
```

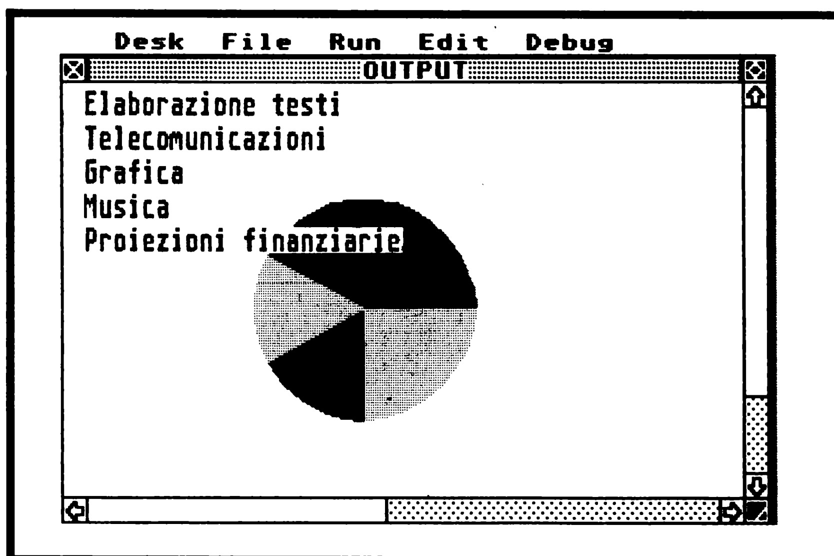


Figura 4.13: diagramma circolare con l'uso dell'istruzione DATA.

```
 90 DATA Telecomunicazioni,5,410,1500
100 DATA Grafica,7,1500,2100
110 DATA Musica,11,2100,2700
120 DATA Proiezioni finanziarie,14,2700,3600
```

Potete usare i vostri dati con il prossimo programma con il quale il diagramma circolare verrà tracciato quando avrete inserito i cinque dati relativi ai nomi e alle rispettive percentuali. La Figura 4.14 vi presenta il diagramma. Accertatevi che la somma delle percentuali sia 100, dato che 100 corrisponde all'intero diagramma. La prima parte del programma specifica le dimensioni delle matrici necessarie, cancella lo schermo ed inizia ad accettare dati:

```
 5 REM Diagramma circolare con percentuali
10 DIM NM$(5),P(10)
20 FULLW 2: CLEARW 2
25 FOR A=1 TO 5
30 INPUT "Inserire nome, percentuale";NM$(A),P(A)
```

Per sapere dove andranno collocati i settori del diagramma circolare, il programma continua ad addizionare i dati in una variabile chiamata SUM. Usando la variabile SUM, il computer crea una nuova variabile nella matrice P(A) in modo da sapere qual è l'angolo iniziale per ciascun settore.

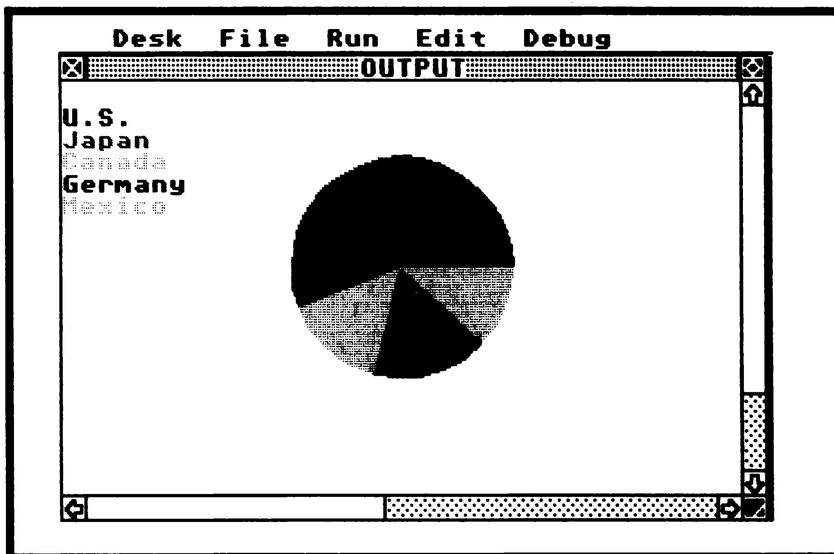


Figura 4.14: diagramma circolare disegnato in base all'inserimento di percentuali.

```
40 SUM = SUM + P(A—1): P(A + 5) = P(A) + SUM
50 REM L'input del loop si chiude con la prossima linea
60 NEXT
```

Una volta che tutti i dati siano stati ricevuti, il computer traccia il grafico, elencando i nomi in concomitanza all'apparire dei settori.

```
 70 CLEARW 2: P(5) = 0: FOR I = 1 TO 5
 80 COLOR I,I,I
 90 PCIRCLE 152,75,50,P(I + 4)*36,P(I + 5)*36
100 REM Nota che gli angoli vanno moltiplicati per 36
110 GOTOXY 0,1: PRINT NM$(I)
120 NEXT
130 GOTO 130
```

L'istruzione ELLIPSE

L'istruzione ELLIPSE può tracciare rispetto a CIRCLE forme più versatili, dato che si possono specificare due raggi, anziché uno solo. Il formato di questa parola chiave è:

ELLIPSE *X,Y,raggio orizzontale,raggio verticale[,angolo iniziale,angolo finale]*

I diversi elementi che si possono usare con ELLIPSE sono teoricamente gli stessi visti con CIRCLE, salvo che si può determinare la forma dell'ellisse specificando i due raggi. Un'ellisse larga e piatta avrà il raggio orizzontale di 100 e il raggio verticale di 10; una alta e stretta potrebbe avere il raggio orizzontale di 12 e il verticale di 50. Potreste anche tracciare una circonferenza specificando due raggi identici, anche se resta più efficiente usare l'istruzione CIRCLE.

Quando usate ELLIPSE, potete intervenire sui due raggi. Modificandone i valori, potrete creare effetti molto interessanti, sia facendoli variare nello stesso senso, sia in senso opposto (aumentando uno quando l'altro diminuisce). Il primo esempio mantiene costante la proporzione dei due raggi per formare un cono ellittico:

```
 5 REM Cono ellittico
10 FULLW 2: CLEARW 2
20 FOR X = 10 TO 300 STEP 4
30 ELLIPSE X,X/1.5,X/2.3,X/4
40 NEXT
50 GOTO 50
```

Fare esperimenti con l'istruzione ELLIPSE può dare dei risultati piuttosto interessanti. Provando diversi valori in un loop FOR...NEXT, potrete affinare delle nuove tecniche grafiche. Quello che segue, per esempio, è un programma molto semplice che, con mia sorpresa, traccia un disegno interessante contenente una losanga:

```
 5 REM Disegno con losanga
10 FULLW 2: CLEARW 2
20 FOR I=15 TO 100 STEP 4
30 ELLIPSE 150,80,I,80—I
40 COLOR 1,RND*15,RND*15: NEXT
50 GOTO 50
```

Nel programma precedente, ho creato un nuovo disegno semplicemente variando il valore di uno dei raggi in senso inverso rispetto all'altro. Quando il raggio orizzontale aumenta, quello verticale diminuisce.

L'istruzione PELLIPSE

Così come si può disegnare un settore di un cerchio specificandone l'angolo iniziale e l'angolo finale, potrete creare settori di ellisse ricorrendo agli stessi argomenti. Inoltre, con l'istruzione PELLIPSE potrete analogamente creare dei settori colorati. Il formato di PELLIPSE è:

PELLIPSE *X,Y,raggio orizzontale,raggio verticale[,angolo iniziale,angolo finale]*

Nel prossimo programma verranno selezionati un angolo iniziale (variabile ANI) e un angolo finale (variabile ANF) casuali e il computer disegnerà il corrispondente settore. Poiché i settori casuali compaiono in rapida successione sullo schermo, ci vorranno solo uno o due secondi perchè compaia un'ellisse piena. Il computer continuerà a tracciare i settori a colori casuali finché non premerete Control-C.

```
 5 REM Settori ellittici casuali
10 FULLW 2: CLEARW 2
20 SAN=INT(RND*36)*100
30 EAN=INT(RND*36)*100
40 PELLIPSE 150,80,100,70,ANI,ANF
50 COLOR 1,RND*15
60 GOTO 20
```

Una significativa differenza tra PELLIPSE e ELLIPSE è che la prima si sovrappone a qualsiasi cosa ci sia già sullo schermo, poiché è una forma piena. Se volete disegnare una serie di ellissi simili ma di colori diversi, una sull'altra, dovrete iniziare con quella più grande e continuare a disegnarle fino alla più piccola.

Se lo faceste nell'ordine contrario, ogni nuova ellisse disegnata coprirebbe la precedente. Potete sfruttare questa caratteristica di sovrapposizione per ottenere effetti grafici.

Il seguente programma traccia un disegno colorato con una losanga al centro:

```
 5 REM Sovrapposizione per creare una forma ellittica
10 FULLW 2: CLEARW 2
20 FOR I=15 TO 100 STEP 4
30 PELLIPSE 150,80,I,80-I
40 COLOR 1,RND*15,RND*15: NEXT
50 GOTO 50
```

Sfruttando la stessa idea, potete scrivere un programma che disegna una croce di ellissi multicolori. La Figura 4.15 ne mostra la forma ma non i colori; per vederli lanciate il programma.

Innanzitutto dovete cancellare lo schermo e iniziare il loop che cambia i valori dei raggi:

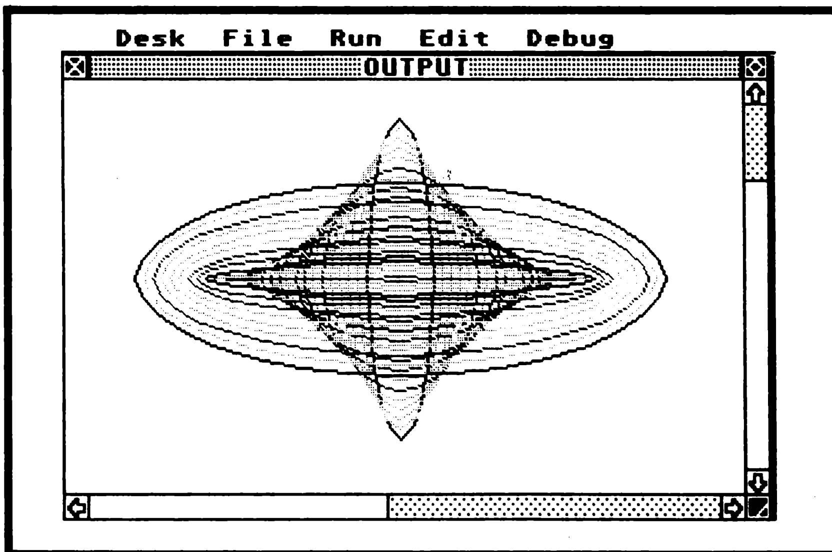


Figura 4.15: croce di ellissi.

```
 5 REM Croce di ellissi
10 FULLW 2: CLEARW 2
20 FOR I=60 TO 10 STEP —5
```

Il secondo loop FOR...NEXT serve a tracciare i quattro distinti disegni. Ogni volta che il loop finisce, il programma legge due numeri nelle variabili X e Y per determinare la localizzazione del centro dell'ellisse. Poi il programma fissa a caso il colore della linea e disegna due ellissi piene: una larga e bassa, l'altra alta e stretta.

```
30 FOR D=1 TO 4
40 READ X,Y: COLOR 1,RND*15
50 PELLIPSE X,Y,I,10
60 PELLIPSE X,Y,10,I
```

Il resto del programma chiude i due loop e contiene le istruzioni DATA. Notate l'istruzione RESTORE (descritta nel Capitolo 2), che viene impiegata ogni volta che finisce il loop *FOR D=1 TO 4.* Dopo che il loop D è completato, ha letto tutte le coordinate X e Y, ma le deve nuovamente usare quando il loop ricomincia. Per riportare il puntatore all'inizio dei dati, usate l'istruzione RESTORE.

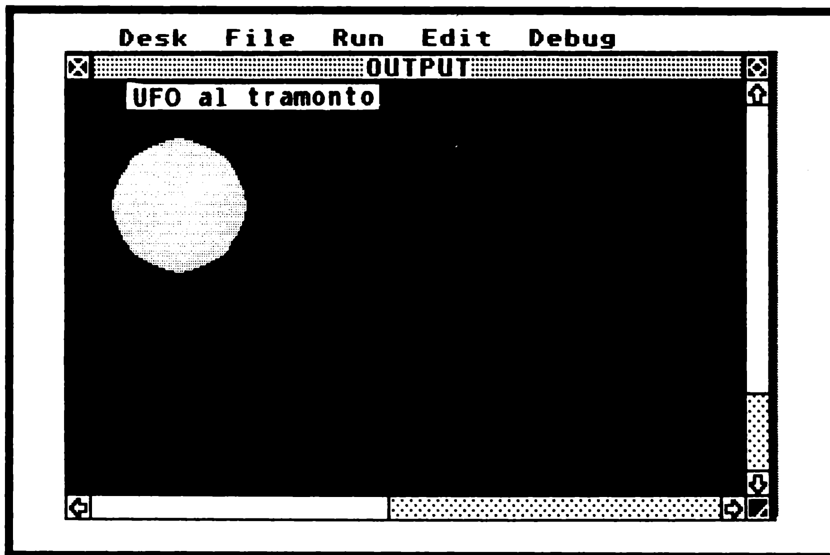


Figura 4.16: output del programma UFO al tramonto.

```
 70 NEXT: RESTORE: NEXT
 80 DATA 60,50
 90 DATA 230,50
100 DATA 80,120
110 DATA 210,120
120 GOTO 120
```

LE PAROLE CHIAVE DELLA GRAFICA IN AZIONE

Tanto per divertirci un po', vediamo ora un programma che impiega tutte le parole chiave che abbiamo descritto finora in questo capitolo. Questo capolavoro, intitolato UFO al tramonto, compare nella Figura 4.16. Il programma inizia cancellando lo schermo e tracciando linee casuali per definire il profilo delle montagne sullo sfondo (vedi il Listato 4.1, linee 10-80).

Una volta disegnate le montagne, supponendo che stiate usando i colori di default, il computer impiega l'istruzione FILL per colorare le montagne di verde, cambia il colore in giallo e disegna il sole con l'istruzione PCIRCLE (linee 90-110)

Poi il computer disegna il corpo dell'UFO con l'istruzione PELLIPSE, le

```
5 REM UFO al tramonto
10 OY=150
20 FULLW 2: CLEARW 2
30 COLOR 1,3,3
40 FOR X=0 TO 300 STEP 40
50 Y=75+RND*50
60 LINEF X,OY,X+40,Y
70 OY=Y
80 NEXT
90 FILL 200,150
100 COLOR 1,14,14
110 PCIRCLE 50,50,30
120 COLOR 1,6,2
130 PELLIPSE 200,50,40,10
140 FOR X=60 TO 90 STEP 5
150 LINEF X+125,50,X+126,50
160 NEXT
170 COLOR 1,5,1: FILL 1,1
180 GOTOXY 3,0: PRINT "UFO al tramonto"
190 GOTO 190
```

Listato 4.1: programma UFO al tramonto.

finestrelle con LINEF e colora il resto del cielo con il blu. Il programma finisce scrivendo UFO al tramonto sullo schermo, ed il loop infinito della linea 190 lascia sullo schermo l'immagine inalterata, finché non premerete Control-C.

COME CAMBIARE I COLORI

Una potente caratteristica dei computer ST, quella di disporre di ben 512 colori diversi, resta spesso sconosciuta anche per i possessori di un ST. Invece non è affatto difficile costruire una propria tavolozza di colori. Anche se dovrete capire alcuni concetti nuovi prima di poter portare colori nuovi sul vostro schermo, non ci vorrà molto tempo per variare ancora di più i vostri programmi grafici adattando alle vostre esigenze i 16 colori disponibili.

Definizione dei colori

Innanzitutto, il computer definisce quali sono i 16 colori della tavolozza in uso per mezzo di numeri registrati nella memoria. Tali numeri rappresentano certe proporzioni dei tre colori fondamentali, rosso, verde e blu, di cui è composto qualsiasi colore. Poiché ogni colore fondamentale ha otto possibili livelli d'intensità, ci sono 8x8x8, cioè 512 colori possibili in tutto. I livelli di rosso, verde e blu sono misurati su una scala da 0 a 7; 0 significa che non è usato nessun colore particolare, mentre 7 significa che il colore in questione è impiegato al suo massimo livello d'intensità. Se in un colore c'e il rosso, ma non il verde né il blu, il colore risulterà naturalmente rosso. Se ad esso aggiungete del verde, il colore cambierà. Se usate la massima intensità di tutti i colori fondamentali, risulta il bianco, mentre l'assenza di qualsiasi colore produce il nero.

Per determinare il numero che produce un certo colore, basta moltiplicare il livello di rosso per 256, il livello di verde per 16 e aggiungere ad essi il livello di blu. Per esempio, se volete il livello 5 di rosso, il livello 3 di verde e il livello 6 di blu, dovrete calcolare il numero del colore come segue:

(256x5)+(16x3)+6=1334

Il numero del vostro colore sarà quindi il 1334.

I 16 numeri che determinano i 16 colori della tavolozza in uso sono conservati in 16 celle di memoria. Il computer ha un numero in memoria, cella 1114, che punta alle celle di questi 16 numeri. Per cambiare i 16 colori in uso, dovrete mettere i 16 numeri nuovi da voi determinati in una matrice e cambiare il puntatore in modo che punti alla cella di memoria dove la vostra matrice comincia.

Per cambiare il puntatore nella cella di memoria 1114, dovrete usare una nuova istruzione, POKE, che serve a mettere un numero in una cella di memoria. Il suo formato è:

POKE *cella di memoria,valore del dato*

In questo caso, la *cella di memoria* è 1114 ed il *valore del dato* è un'altra cella di memoria, il punto dove la matrice inizia. Per scoprire dove inizia la matrice, usate VARPTR (puntatore di variabile), che restituisce l'indirizzo di una variabile in memoria.

Il prossimo programma in primo luogo imposta una matrice A%, che memorizza dei colori casuali nelle sue 15 celle. Notate che l'espressione della linea 40 impiega la formula:

(256 × *livello di rosso*)+(16 × *livello di verde*)+(*livello di blu*)

per calcolare i numeri dei colori.

```
 5 REM Memorizzare colori casuali
10 FULLW 2: CLEARW 2
20 DIM A%(15)
30 FOR I=0 TO 15
40 A%(I)=256*INT(RND*7)+16*INT(RND*7)+RND*7: NEXT
```

Poi il programma registra la cella di memoria 1114 nella variabile a precisione doppia N (ricordate di usare DEFDBL per stabilire che questa variabile è a precisione doppia quando cambiate i colori). L'istruzione POKE mette il puntatore di variabile nella cella di memoria 1114 e sullo schermo viene stampata la parola *Colori!*. Infine il programma ritorna alla linea 30 e può così selezionare ancora colori casuali per far lampeggiare velocemente lo schermo man mano che passa a diverse tavolozze.

```
50 DEFDBL N: N=1114: POKE N,VARPTR(A%(0))
60 PRINT "Colori!":GOTO 30
```

Numeri di colore

Una complicazione che insorge cambiando i colori è che i numeri di colore nella matrice della tavolozza **non** corrispondono ai numeri di colore del BASIC ST. Per esempio, quando battete *COLOR 2,3*, cambiate il colore del testo a 2 ed il colore di linea a 3; questi sono i numeri di colore del BASIC ST.

Numero del colore in BASIC ST	Posizione nella matrice della tavolozza	Colore
0	0	bianco
1	15	nero
2	1	rosso
3	2	verde
4	4	azzurro
5	6	blu
6	3	rosso scuro
7	5	verde scuro
8	7	grigio
9	8	grigio scuro
10	9	ottanio
11	10	ottanio scuro
12	12	magenta
13	14	magenta scuro
14	11	giallo
15	13	giallo scuro

Tabella 4.2: tabella di conversione dei colori.

Tuttavia, se volete cambiare i colori nei registri 2 e 3 per produrre un effetto diverso, non basta cambiare A%(2) e A%(3) nella matrice della tavolozza. Dovrete invece ricorrere alla tabella di conversione nella Tabella 4.2 per scoprire quali variabili della matrice dovrete modificare per ottenere lo stesso cambiamento in un colore BASIC ST.

Supponiamo che vogliate cambiare il colore BASIC ST 13 (magenta scuro) in un colore diverso. Dovrete modificare la variabile A%(14), o qualsiasi nome abbia la matrice della tavolozza, poi usare la funzione VARPTR per fare in modo che il computer consideri quella matrice come sua tavolozza ed emetta un'istruzione COLOR, del tipo *COLOR 13,13* per mettere il vostro nuovo colore dove prima c'era il magenta scuro.

È anche più interessante cambiare i colori quano lo schermo visualizza qualche immagine. Il prossimo programma disegna una serie di cerchi concentrici pieni e poi inizia ad alterare la tavolozza. Lanciate il programma e guardate cosa succede.

```
 5 REM Cerchi concentrici pieni
10 FULLW 2: CLEARW 2
20 FOR R=100 TO 10 STEP —10
```

```
30 COLOR 1,R/10,R/10: PCIRCLE 150,75,R
40 NEXT
50 DIM A%(15)
60 FOR I=0 TO 10
70 A%(I)=256*INT(RND*7)+16*INT(RND*7)+RND*7: NEXT
80 DEFDBL N: N=1114: POKE N,VARPTR(A%(0))
90 GOTO 60
```

È necessario ridefinire l'intera tavolozza di 16 colori quando assegnate un nuovo set di colori, ma una volta che avrete fatto ciò, potrete anche cambiare un solo colore. Nel prossimo programma, solo lo sfondo dello schermo lampeggia di colori casuali:

```
 5 REM Sfondo lampeggiante a diversi colori
10 DIM A%(16): DEFDBL N: N=1114
20 FULLW 2: CLEARW 2
30 FOR I=0 TO 14 STEP 2
40 A%(I)=1298
50 A%(I+1)=1889
60 NEXT
70 POKE N,VARPTR(A%(0))
80 A%(0)=INT(RND*7)*256+INT(RND*7)*16+INT(RND*7)
90 GOTO 70
```

Basta invece una piccola modifica al programma per far lampeggiare l'immagine in primo piano:

```
 5 REM Immagine lampeggiante a diversi colori
10 DIM A%(16): DEFDBL N: N=1114
20 FULLW 2: CLEARW 2
30 FOR I=0 TO 14 STEP 2
40 A%(I)=1298
50 A%(I+1)=1889
60 NEXT
70 POKE N,VARPTR(A%(0))
80 A%(14)=INT(RND*7)*256+INT(RND*7)*16+INT(RND*7)
90 GOTO 70
```

Infine, se volete vedere l'intero set di 512 colori, provate il prossimo programma, che ripercorre ciclicamente l'intera sequenza. Il programma visualizzerà i livelli di rosso, di verde e di blu nella parte alta dello schermo, man

mano che il cerchio pieno cambia colore. Se vedete dei colori che vorreste impiegare nel vostro programma, potete trascriverne il numero.

```
  5 REM Visualizza 512 colori
 10 DIM A%(16): DEFDBL N: N=1114
 20 FULLW 2: CLEARW 2
 30 FOR I=0 TO 14 STEP 2
 40 A%(I)=1298
 50 A%(I+1)=1889
 60 NEXT
 70 POKE N,VARPTR(A%(0))
 80 COLOR 1,2,2: PCIRCLE 150,80,50
 90 FOR A=1 TO 7
100 FOR B=1 TO 7
110 FOR C=1 TO 7
120 A%(1)=A*256+B*16+C
130 POKE N,VARPTR(A%(0))
140 GOTOXY 0,0: PRINT A,B,C
150 NEXT: NEXT: NEXT
```

Cambiare i colori è particolarmente utile quando si disegnano immagini a tutto schermo o si raffigurano oggetti reali. Le due cose più importanti da ricordare sono di puntare il computer alla matrice di variabili giusta e di usare la tabella di conversione dei colori (Tabella 4.2) per essere sicuri di cambiare il colore giusto.

COME DISEGNARE IN PROSPETTIVA TRIDIMENSIONALE

Il BASIC ST non ha comandi speciali per disegnare oggetti tridimensionali, ma potete disegnare cubi, piramidi ed immagini con effetto di profondità senza troppi problemi. Per rendere la prospettiva, dovete far diventare sempre più piccole le linee (o la forma) che costituiscono un oggetto; come nella realtà fisica, le cose vicine sembrano più grandi e quelle lontane sembrano più piccole. Se le linee si allontanano al punto da scomparire su un'immaginaria linea di orizzonte, potete farle convergere in un punto; nel disegno prospettico, il punto dell'orizzonte su cui convergono tutte le linee si chiama *punto di fuga*, essendo questo il punto in cui le linee che vi convergono si sottraggono (sfuggono) alla vista.

Il prossimo programma disegna un quadrato con quattro linee che partono

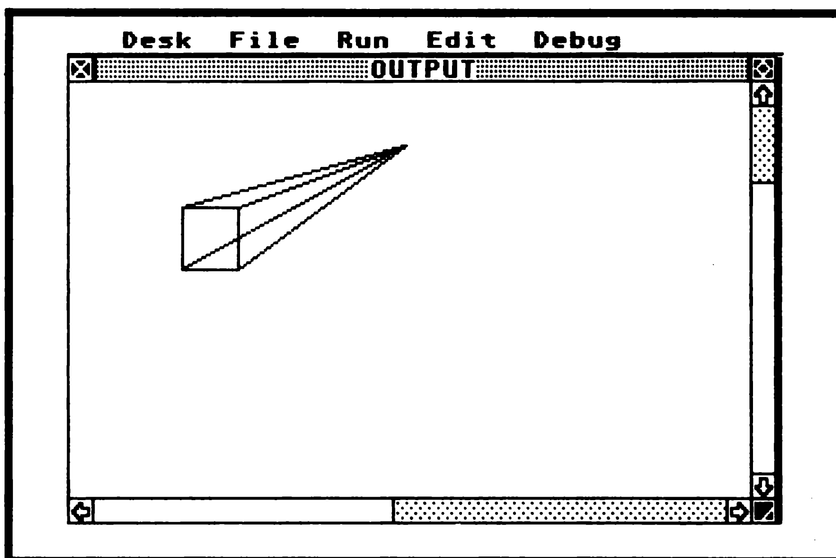


Figura 4.17: quadrato con punto di fuga.

dagli angoli e si allontanano verso l'orizzonte. Le linee convergono al punto di fuga per dare all'immagine l'illusione della profondità. Il risultato è quello della Figura 4.17

```
  5 REM Quadrato con punto di fuga
 10 FULLW 2: CLEARW 2
 20 LINEF 50,50,75,50
 30 LINEF 75,50,75,75
 40 LINEF 75,75,50,75
 50 LINEF 50,75,50,50
 60 FOR I=1 TO 4
 70 READ X,Y
 80 LINEF X,Y,150,25
 90 NEXT
100 DATA 50,50,75,50
110 DATA 75,75,50,75
120 GOTO 120
```

Per mettere le linee nella posizione appropriata basta usare come origine di ciascuna linea le stesse coordinate impiegate per i vertici del quadrato.

Il prossimo programma disegna la stessa forma, ma sposta il punto di fuga in basso a destra tracciando così la stessa prospettiva 15 volte (in 15 colori).

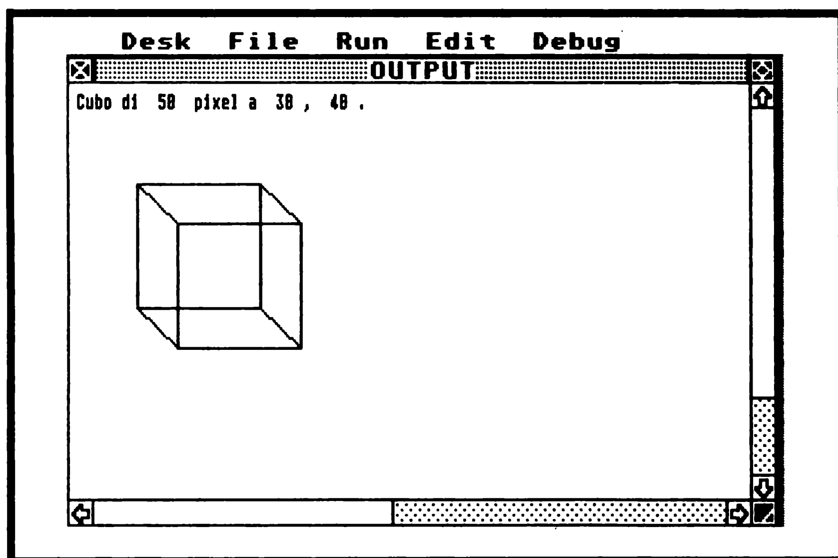


Figura 4.18: output del programma Cubo.

Come vedrete, anche se il punto di fuga cambia, le immagini hanno tutte uno spessore.

```
  5 REM Punto di fuga con i colori
 10 FULLW 2: CLEARW 2
 20 LINEF 50,50,75,50
 30 LINEF 75,50,75,75
 40 LINEF 75,75,50,75
 50 LINEF 50,75,50,50
 55 FOR C=1 TO 15: COLOR 1,C,C
 60 FOR I=1 TO 4
 70 READ X,Y
 80 LINEF X,Y,150+C*7,25+C*6
 90 NEXT: RESTORE: NEXT
100 DATA 50,50,75,50
110 DATA 75,75,50,75
120 GOTO 120
```

L'ultimo programma di questo capitolo riportato nel Listato 4.2, è diverso non solo perché svolge una specifica funzione - disegna un cubo di posizione, dimensioni e colore che voi specificherete - ma visualizza anche le linee in BASIC ST necessarie per creare lo stesso cubo in un programma, come mostra la Figura 4.18. Per ottenere lo stesso risultato, potete copiare le linee diretta-

```
5 REM Cubo
10 FULLW 2:CLEARW 2: COLOR 1,1,1
20 PRINT "Cubo": PRINT
30 INPUT "Di che dimensioni deve essere il cubo";SZ:
XSZ=INT(SZ*1.1)
40 PRINT "Dove va collocato il cubo?":
INPUT "Inserisci X,Y"; X,Y: OX=X: OY=Y
50 PRINT "Di che colore deve essere il cubo?":
INPUT "Inserisci un numero da 0 a 15"; CL
60 CLEARW 2: COLOR 1,CL,CL: GOSUB BOX:
X=X+XSZ/3: Y=Y+SZ/3: GOSUB BOX: X=X-XSZ/3
70 Y=Y-SZ/3
80 LINEF X,Y,X+XSZ/3,Y+SZ/3
90 LINEF X+XSZ,Y,X+XSZ*4/3,Y+SZ/3
100 LINEF X,Y+SZ,X+XSZ/3,Y+SZ*4/3
110 LINEF X+XSZ,Y+SZ,X+XSZ'4/3,Y+SZ*4/3
120 GOTOXY 0,0: PRINT "Cubo di "SZ" pixel a "X", "Y"."
130 GOTO INSTRUCT
140 BOX: LINEF X,Y,X+XSZ/Y
150 LINEF X,Y,X,Y+SZ: LINEF X,Y+SZ,X+XSZ,Y+SZ
160 LINEF X+XSZ,Y,X+XSZ,Y+SZ
170 RETURN
180 INSTRUCT: FOR D=1 TO 5000: NEXT: CLEARW 2
190 PRINT "Le linee necessarie per creare"
200 PRINT "questo cubo sono:"
210 PRINT: PRINT "COLOR 1 ,"CL","CL"
220 PRINT "LINEF"X","Y","X+XSZ","Y
230 PRINT "LINEF"X","Y","X","Y+SZ
240 PRINT "LINEF"X","Y+SZ","X+SZ","Y+SZ
250 PRINT "LINEF"X+XSZ","Y","X+XSZ","Y+SZ
260 X=INT (X=XSZ/3): Y=INT (Y+SZ/3):
IF FL=0 THEN FL=1: GOTO 220 ELSE X=OX: Y=OY
270 PRINT "LINEF"X","Y","INT(X+SZ/3)","INT(Y+SZ/3)
280 PRINT "LINEF"X+XSZ","Y","INT(X+XSZ*4/3)","INT(Y+SZ/3)
290 PRINT "LINEF"X","Y+SZ","INT(X+XSZ/3)","INT(Y+SZ*4/3)
300 PRINT "LINEF"X+XSZ","Y+SZ","INT(X+XSZ*4/3)","INT(Y+SZ*4/3)
310 GOTO 310
```

Listato 4.2: programma Cubo.

mente dallo schermo ed usarle una per una, oppure nel contesto di un programma più grande. La prima parte del programma (linee 10-50) predispone lo schermo e accetta dimensione, posizione e colore dalla tastiera. La variabile della dimensione, SZ, viene moltiplicata per 1.1 nella linea 30 per rendere la larghezza leggermente maggiore dell'altezza (poiché i pixel non sono perfettamente quadrati).

L'ST cancella lo schermo, imposta il colore e va alla subroutine BOX, che disegna sullo schermo il primo quadrato (linea 60). I valori delle variabili X e Y vengono poi alterati per tracciare l'altro quadrato in una certa posizione rispetto al primo. La subroutine BOX viene impiegata ancora una volta e le variabili X e Y vengono riportate al loro valore originale (linea 70).

Ora il programma collega i due quadrati con quattro comandi LINEF (linee 80-110). Le coordinate di queste linee vengono determinate con le variabi-

li X e Y, associate alle variabili XSZ e SZ, che determinano le dimensioni del cubo.

Nelle linee 120-130, il programma porta il cursore nella sua posizione naturale (posizione 0,0) e stampa le dimensioni e la posizione del cubo misurate in pixel. Per visualizzare le linee che servono per ricreare il cubo, il programma salta alla routine chiamata INSTRUCT.

Nelle linee 140-170, la subroutine BOX, a cui il programma accede due volte, disegna un quadrato basato sulle variabili X e Y (per la posizione) e XSZ, SZ (per determinare le dimensioni). Tracciare il quadrato è semplice, perché basta solo specificare le coordinate degli angoli.

La subroutine INSTRUCT (linea 180) inizia facendo fermare il programma per alcuni istanti prima che lo schermo venga ripulito, per darvi modo di vedere il cubo che avete creato. Poi le linee del programma vengono visualizzate sullo schermo; esse vengono semplicemente copiate dal resto del programma, ma dato che i valori delle variabili possono essere visualizzati direttamente, compariranno i numeri (per esempio 20, 45 e 12) anziché i nomi delle variabili. Questo vi permette di riprodurre lo stesso cubo in un programma diverso, dato che non è necessario riprogrammare le variabili X, Y, XSZ e SZ; le istruzioni LINEF riportano già i numeri.

Potreste pensare anche ad altri programmi ausiliari che vi aiutino nella programmazione. L'uso del mouse (vedi Capitolo 7) vi aiuterà a disegnare e riprodurre grafici con ancora maggiore facilità.

COME USARE LA GRAFICA

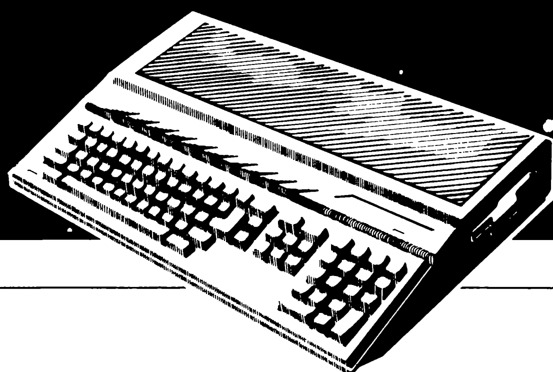
In questo capitolo ho trattato un'area piuttosto vasta e forse vorrete lasciare da parte il libro per qualche ora (o qualche giorno) per esercitare le vostre nuove conoscenze di grafica. La grafica può essere molto divertente, ma può anche essere alla base di applicazioni serie. Con LINEF, CIRCLE, PCIRCLE, ELLIPSE, PELLIPSE, FILL e la possibilità di cambiare la tavolozza dei 16 colori, ci sono decine di modi interessanti per sfruttare la grafica dell'ST. Vediamone alcuni:

- *Grafici statistici*, quali grafici lineari, a istogrammi e diagrammi circolari. Potreste anche scrivere dei programmi per tracciare organigrammi, la piantina di un ufficio, di un impianto e il diagramma di un prodotto. Poiché la grafica può operare congiuntamente a dati numerici, le applicazioni commerciali sono semplici da scrivere e sono un modo eccellente di presentare le informazioni. Tratteremo ancora l'uso dei dati numerici in associazione con la grafica nel Capitolo 8.

- *Programmi educativi*, per insegnare geometria, tracciare grafici per analisi e trigonometria, o magari usare grandi grafici colorati per insegnare ad alunni delle elementari le addizioni e le sottrazioni. La grafica rende interessanti i programmi soprattutto per i bambini e se i più giovani in famiglia vogliono imparare qualcosa, potrete sempre scrivere dei programmi che li aiutino nell'apprendimento. Meglio ancora, se i bambini se la cavano con la programmazione, dovranno approfondire gli argomenti per poter scrivere da sé i programmi. In questo modo siete sicuri che capiscano a fondo le soluzioni dei problemi.

- *Videogiochi*, che sono l'uso più ovvio della grafica a colori. I videogiochi sono in effetti tra i programmi più difficili da scrivere, ma se volete provare a sciverne uno abbastanza semplice, la grafica dell'Atari renderà il tentativo molto più interessante.

La grafica dell'ST è pratica, facile da usare e molto divertente. Come sempre, il miglior insegnante è la vostra esperienza, quindi provate a scrivere qualche programma (diciamo una decina) usando ciascuna delle parole chiave. Imparerete molto più in fretta, e scoprirete che programmare in BASIC ST con la grafica può essere uno dei modi più simpatici di conoscere più a fondo le possibilità del vostro ST.



SUONO E MUSICA

La capacità dell'ST di produrre suoni e musica è potente e utile quanto le sue potenzialità grafiche. Potete creare pezzi musicali ed effetti sonori e potete usare fino a tre canali diversi contemporanemente. Poiché è fornito di un potente chip acustico AY-3-8910 della General Instrument Corporation, produrre musica e suoni è molto più facile sull'ST che su molti altri computer .

Questo chip è particolarmente versatile:

- Può produrre sia suoni che musica.
- Potete creare 4096 suoni diversi con l'istruzione SOUND.
- Potete modificare le caratteristiche di un suono usando l'istruzione WAVE.
- Potete produrre fino a tre voci contemporaneamente; una *voce* o canale, è il termine usato per descrivere il suono che si sta producendo. Quando producete tre note o suoni simultaneamente (per esempio per formare un accordo), significa che state usando tre voci; naturalmente potete anche usarne solo una o due.

Anche se il BASIC ST prevede solo due parole chiave per suono e musica, SOUND e WAVE, potete generare una vasta gamma di toni con i programmi.

L'ISTRUZIONE SOUND

L'istruzione che userete più spesso sia per i suoni che per la musica è SOUND. Il suo formato è:

SOUND *voce,volume,nota,ottava,durata*

Vediamo una breve descrizione di ciascun argomento.

Argomenti di SOUND

Voce Si emette il suono attraverso un canale. Il valore che identifica il canale può variare da 1 a 3. Usando più canali contemporaneamente, potete produrre accordi ed effetti sonori particolari. Per usare canali multipli, tuttavia, dovete prima imparare l'uso dell'istruzione WAVE, descritta più avanti in questo capitolo.

Volume Il numero che rappresenta il volume può variare da 0 (silenzio) a 15 (volume massimo). Potete usare qualsiasi numero intero in questo inter-

vallo per dire al computer quanto deve essere intenso il suono, ma probabilmente userete quasi sempre 15.

Nota Ci sono 12 note nella scala musicale e l'argomento nota dice al computer quale nota suonare; la tabella 5.1 riporta le note ed i numeri corrispondenti. Una nota *diesis* è più alta di mezzo tono rispetto alla precedente, La diesis, o La#, si trova a metà tra La e Si. Notate che più alto è il numero più acuto è il suono; questo sistema di note e numeri è particolarmente pratico quando programmate una canzone al computer direttamente dallo spartito. Notate che nella notazione musicale La diesis si scrive La#.

Ottava L'ottava vi offre una notevole flessibilità nello stabilire quanto alto o basso il suono deve essere. L'ottava può variare da 1 (note profonde, di basso) a 8 (note molto acute). Combinando note ed ottave diverse, avrete un totale di 96 note tra cui scegliere. La Figura 5.1 illustra un'ottava della tastiera di un pianoforte.

Durata Infine, l'argomento durata dell'istruzione SOUND dice al computer per quanto tempo deve emettere un suono. Può variare da 0 a 65535 e ciascuna unità di durata è un quindicesimo di secondo. Quindi una durata pari a 150, tiene un suono per tre secondi, mentre la durata 0 emette il suono per una frazione di secondo. In effetti la durata 0 non corrisponde a 0 secondi.

Numero	Nota	Numero	Nota
1	Do	7	Fa diesis
2	Do diesis	8	Sol
3	Re	9	Sol diesis
4	Re diesis	10	La
5	Mi	11	La diesis
6	Fa	12	Si

Tabella 5.1: note musicali e numeri corrispondenti.

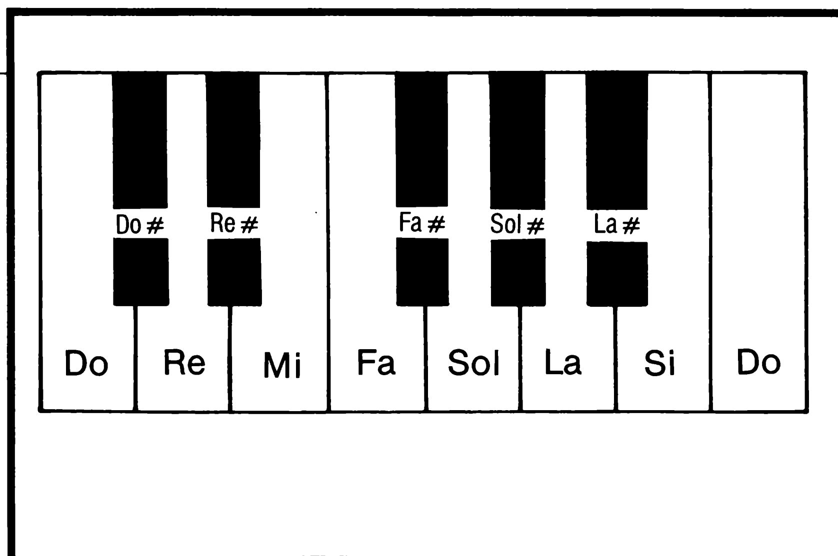


Figura 5.1: le note della tastiera del pianoforte.

È il tempo che il BASIC impiega ad eseguire le istruzioni che precedono la successiva istruzione WAVE o SOUND. Se non ci sono altri comandi dopo SOUND, la nota continuerà all'infinito; l'argomento durata ha significato solo nel contesto di un programma, dove qualche altro comando segue SOUND. Per produrre effetti sonori anziché note musicali, è opportuno usare un numero molto basso per la durata (per esempio 0), in modo tale che i suoni si fondano tra loro permettendovi una notevole flessibilità nel creare effetti sonori.

Esempi d'istruzioni SOUND

Il programma che segue impiega alcune istruzioni SOUND, per mostrarvi come gli argomenti funzionino.

```
10 WAVE 0
20 SOUND 1,15,5,3,50
30 WAVE 2
40 SOUND 3,5,9,5,25
50 WAVE 1
60 SOUND 2,0,1,4,100
```

La prima istruzione SOUND emette la nota Mi (5) nell'ottava 3 al volume

massimo (15), attraverso il canale 1, per la durata di un secondo e la seconda emette la nota Sol # nell'ottava 5 per mezzo secondo dal canale 3 al volume 5. La terza emette la nota Do nell'ottava 4 per due secondi dal canale 2; tuttavia, siccome il volume è 0, il suono non sarà udibile.

Notate che ciascuna istruzione SOUND è preceduta da un'istruzione WAVE. Imparerete in che modo queste due istruzioni operino insieme più avanti in questo capitolo.

Cambiare gli argomenti dell'istruzione SOUND è un modo facile di mutare le caratteristiche dei suoni e della musica che produrrete. Per esempio, potreste scrivere un programma come il prossimo, che ripetutamente cambia il volume dal livello minimo al livello massimo.

```
 3 REM Cambiare volume
 5 WAVE 1
10 FOR V=0 TO 15
20 SOUND 1,V,1,4,0
30 NEXT
40 GOTO 10
```

(Naturalmente potete cambiare il volume anche regolando la manopolina del monitor. I monitor Atari sono dotati di altoparlamte interno; girando la manopolina a sinistra, potete aumentare il suono e girandola verso destra si diminuisce il suono. Sentirete un piccolo scatto nella manopolina, in corrispondenza del quale si ottiene di solito un volume equilibrato).

COME LAVORARE IN BINARIO

Poter cambiare il volume e l'altezza di un suono, tuttavia, non basta a darvi la versatilità necessaria per produrre suoni più complessi. L'istruzione WAVE vi permette di modificare radicalmente le caratteristiche dei suoni prodotti dal vostro ST.

Prima di descrivere l'istruzione WAVE nei dettagli, è necessario che comprendiate alcuni fondamenti dell'aritmetica binaria. Innanzitutto, considerate il nostro normale sistema di contare: il sistema decimale. Quando guardate un numero, quale 2201, capite immediatamente quale è il suo valore perché da sempre avete usato i numeri decimali.

Se tuttavia analizziamo questo numero, diventano facilmente comprensibili anche altri sistemi numerici. Per definizione, il sistema decimale si basa su unità di dieci. Cominciando da destra, il primo posto di un numero rappresenta le unità (10^0 - o N^0 - è uguale a 1), il secondo rappresenta le decine (10^1 è uguale a 10 - ovvero N^1 è uguale a N), La terza rappresenta le centinaia (10^2

è uguale a 10×10, ovvero 100) e così via. Potreste così analizzare 2201 dicendo che:

```
2201
││││
│││└──1 × 10^0 =     1
││└───0 × 10^1 =     0
│└────2 × 10^2 =   200
└─────2 × 10^3 =  2000
```

2201	$1 \times 10^0 =$	1
	$0 \times 10^1 =$	0
	$2 \times 10^2 =$	200
	$2 \times 10^3 =$	2000

Sommando questi numeri - 2000, 200, 0 e 1 - si ottiene 2201.

Il sistema binario è simile, salvo che si basa sulle potenze di 2, anziché di 10, come risulta dalla Tabella 5.2.

Le uniche cifre di un numero binario sono 0 o 1; per esempio, 0110101, 1110101 o 11111111010101. Potete determinare il valore decimale di ciascuno di questi numeri moltiplicando ciascuna cifra per il valore che il posto corrispondente rappresenta e poi sommando i risultati. Calcoliamo il valore decimale di 11010:

11010		
	$0 \times 2^0 =$	0
	$1 \times 2^1 =$	2
	$0 \times 2^2 =$	0
	$1 \times 2^3 =$	8
	$1 \times 2^4 =$	16
	Totale =	26

Il numero 11010, quindi, corrisponde al numero decimale 26. C'è da aggiungere che ciascun posto nel numero binario si chiama *bit* (abbreviazione di binary digit, cioè cifra decimale). Quindi si dice che il numero 11010 è composta da cinque bit.

I numeri binari sono molto importanti perché tutte le operazioni del computer sono rappresentate e svolte in un sistema di due soli possibili stati: acceso/spento, vero/falso. Supponiamo che il computer tenga un "registro" delle condizioni verificate o no, come:

Bit 0: La stampante è accesa (imposta il bit a 1) o spenta (imposta il bit a 0)?
Bit 1: Il mio utente è Giovanni (1) o Susanna (0)?
Bit 2: È giorno (1) o sera (0)?
Bit 3: È stato copiato o cancellato qualche file (1) o nessuno (0)?
Bit 4: Il monitor è monocromatico (1) o a colori (0)?
Bit 5: È un giorno lavorativo (1) o di fine settimana (0)?

Usando queste condizioni, il numero binario 100011 (decimale 35) vi dice

Posizione del numero da destra	Posto binario	Valore del posto
1	2^0	1
2	2^1	2
3	2^2	4
4	2^3	8
5	2^4	16
6	2^5	32
7	2^6	64
8	2^7	128

Tabella 5.2: valori dei posti binari.

che la stampante è accesa, che Giovanni sta usando il computer, che è sera, che nessun file è stato copiato o cancellato, il computer impiega uno schermo monocromatico ed è un giorno lavorativo. Usando il numero decimale 35 potreste comunicare le stesse informazioni. In effetti con un solo numero potreste avere 50 diverse condizioni e rispondere a ciascuna di esse sì o no.

Un computer ST lavora con un determinato numero di bit per gestire le informazioni: otto. Questi otto bit formano un *byte.* Impostando i bit di un byte potrete specificare le condizioni esatte che il computer deve seguire. Il valore di un byte può variare da 00000000 (0 decimale) a 11111111 (255 decimale), dando in tutto 256 combinazioni possibili. Per ottenere il corrispondente valore decimale, sommate il valore dei posti corrispondenti a 1, leggendo dalla cifra più a destra a quella più a sinistra.

Gli argomenti dell'istruzione WAVE si basano su semplici numeri binari.

L'ISTRUZIONE WAVE

Potete usare l'istruzione WAVE per produrre effetti sonori, alterare i suoni che produce l'istruzione SOUND e aprire canali specifici da cui far provenire il suono. Il formato dell'istruzione WAVE è:

WAVE *attivatore[,busta,forma,periodo,ritardo]*

Segue la descrizione di ciascun argomento.

Argomenti di WAVE

Attivatore Con l'argomento attivatore potete specificare quali voci volete usare per la musica e quali per i rumori. Quando si usa una voce per i rumori, risulta un diverso tipo di suono. La forma d'onda di una nota musicale è liscia e continua; quella di un rumore è discontinua. I bit 0, 1 e 2 dicono al computer di emettere la musica attraverso il primo, il secondo o il terzo canale, rispettivamente. I bit 3, 4 e 5 dicono al computer di emettere i rumori attraverso il primo, il secondo o il terzo canale, rispettivamente (i bit 6 e 7 non si usano). Potete *fissare* a 1 quanti bit volete; per esempio, se volete che la voce 1 emetta musica mentre la voce 2 e 3 emettono rumori, il corrispondente numero binario sarà 00110001, che si traduce nel decimale 49. Dovrete convertire i binari in decimali, poiché WAVE (come le altre parole chiave BASIC ST) non accetta numeri binari.

Busta L'argomento busta dice al computer quali voci devono essere interessate dall'istruzione WAVE. Ciò è rilevante nel caso in cui, per esempio, vo-

Numero	Canali interessati
1	1
2	2
3	1 e 2
4	3
5	1 e 3
6	2 e 3
7	1, 2 e 3

Tabella 5.3: valori dell'argomento busta.

gliate che solo uno o due canali abbiano le caratteristiche che stabilite con l'istruzione WAVE e che i rimanenti rimangano inalterati. Il bit 0 indica che volete che WAVE interessi la voce 1, il bit 1 interessa la voce 2 e il bit 2 interessa la voce 3. Per alleggerire in parte il lavoro di decodifica, la Tabella 5.3 elenca i numeri decimali da specificare per mettere in relazione l'istruzione WAVE con certi canali.

Forma Una *forma d'onda* è un modello programmato nel computer per fare in modo che il volume di un suono venga modulato in un certo modo. Per cambiare velocemente il volume di un suono, da alto a basso, per ottenere un effetto tremolo, usate una forma d'onda frastagliata. La Figura 5.2 mostra le otto forme d'onda disponibili ed i numeri corrispondenti, che dovrete usare nell'argomento forma.

Periodo Potete cambiare l'equilibrio tra toni bassi e toni alti di un suono regolando il valore del periodo. Questo argomento varia da 0 (prevalenza dei toni acuti) a 31 (prevalenza dei toni bassi). L'argomento periodo essenzialmente stabilisce con che velocità il suono cicla; verso la fine di questo capitolo, imparerete ad usare numeri più grandi per scopi diversi dal controllo dell'equilibrio tra alti e bassi.

Ritardo L'argomento ritardo stabilisce per quanto tempo un suono va emesso prima che il computer passi al suono successivo. Come nell'istruzione SOUND,

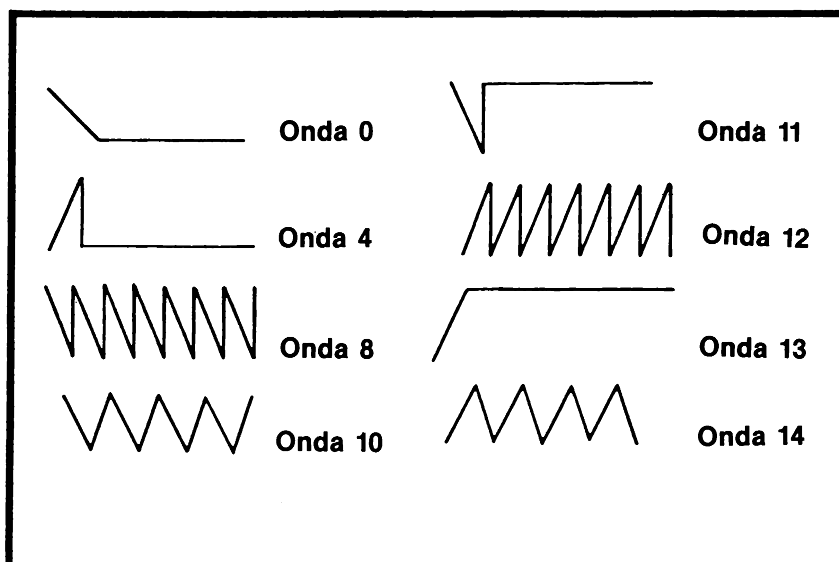


Figura 5.2: forme d'onda.

potete fissarlo con incrementi di un quindicesimo di secondo, il chè significa che 250 tiene un suono per cinque secondi. Il massimo ritardo possibile è 65,535.

Usi di WAVE

L'istruzione WAVE è utile anche per aumentare i tipi di musica e di effetti sonori che potete produrre. Potete aprire e chiudere diversi canali per musica ed effetti sonori a una, due o tre voci e con WAVE 0, potete disattivare completamente i canali sonori.

COME USARE SOUND E WAVE

L'istruzione SOUND offre 96 note tra cui scegliere; l'istruzione WAVE porta il totale a 4,096.

Note musicali

L'uso più ovvio dell'istruzione SOUND è produrre musica. Poiché le frequenze delle 96 diverse note sono già registrate nella memoria del computer, non c'è bisogno di eseguire difficili conversioni di numeri per calcolare quale frequenza una certa nota richieda.

Questo primo esempio di programma visualizza ciascuna nota e l'emette nella prima voce. Prima di tutto, cancellate la finestra Output e inserite l'audio con l'istruzione WAVE 1:

```
 5 REM Visualizzare ed emettere note
10 FULLW 2: CLEARW 2
20 WAVE 1
```

Ora il programma ricorre ad un loop FOR...NEXT per leggere tutte le 12 note; man mano che viene letta una nota, il computer la visualizza sullo schermo e la emette usando i valori del loop FOR...NEXT:

```
30 FOR N=1 TO 12
40 READ N$
50 PRINT "Questa nota e' un ";N$
55 SOUND 1,15,N,5,25
```

```
60 NEXT
70 DATA DO,DO#,RE,RE#,MI,FA,FA#,SOL,SOL#,LA,LA#,SI
80 WAVE 0
RUN
```

Questa nota e' un DO
Questa nota e' un DO#
Questa nota e' un RE
Questa nota e' un RE#
Questa nota e' un MI
Questa nota e' un FA
Questa nota e' un FA#
Questa nota e' un SOL
Questa nota e' un SOL#
Questa nota e' un LA
Questa nota e' un LA#
Questa nota e' un SI

Quando date un'istruzione SOUND, quella nota continuerà ad essere emessa finché non incontra l'istruzione SOUND successiva o finché WAVE 0 (come nella linea 80) disattiva la voce attraverso la quale il suono veniva emesso. Un modo alternativo d'interrompere un suono è di dare un'altra istruzione SOUND dallo stesso canale, ma a volume 0.

```
5 REM Programma Scala musicale
10 FULLW 2: CLEARW 2: WAVE 0: COLOR 1,1,1
20 FOR Y=20 TO 60 STEP 10
30 LINEF 10,Y,500,Y: NEXT
40 FOR X=10 TO 15: LINEF X,20,X,60: NEXT
50 WAVE 1
60 FOR L=1 TO 2: READ A,B,C
70 FOR N=A TO B STEP C: READ NT,O
80 PCIRCLE (L*80)+N*25,75-(N*5),10
90 LINEF (L*80)+N*25+10,75-(N*5),(L*80)+N*25+10,55-(N*5)
100 SOUND 1,15,N,O,30
110 NEXT: NEXT: WAVE 0
120 DATA 1,8,1
130 DATA 1,3,3,3,5,3,6,3,8,3,10,3,12,3,1,4
140 DATA 8,1,-1
150 DATA 1,4,12,3,10,3,8,3,6,3,5,3,3,3,1,3
```

Listato 5.1: programma Scala musicale.

Un modo più elegante di mostrare le note è quello di ricorrere alla grafica per disegnare le note su un pentagramma man mano che vengono emesse. Il programma del Listato 5.1 segue proprio questo procedimento, tracciando dapprima il pentagramma usato per la notazione musicale. Le cinque linee orizzontali sono disegnate dalle linee 20 e 30. Poi le barre più scura all'inizio e alla fine del pentagramma vengono tracciate dalla linea 40 con un altro loop FOR...NEXT e un'istruzione LINEF.

Perché il programma suoni le note della scala prima in senso ascendente, poi nel senso opposto, impostate il loop della linea 60. Il loop conta da 1 a 2 in modo tale che la scala viene suonata due volte e la variabile nel loop determina dove la nota va disegnata sullo schermo. Sempre nella linea 60, il programma leggerà le variabili A, B e C in memoria per scoprire in che punto deve cominciare il loop successivo (ad A) e dove deve finire (in B), nonché di quale incremento il loop deve procedere (C). Altri due elementi di dati vengono letti nella linea 70: la nota e l'ottava in cui deve essere suonata.

Nella linea 80, il computer disegna le note sul pentagramma. Le variabili L (usata nel primo loop) e N (il numero della nota) mettono le note nella posizione corretta. L'istruzione PCIRCLE traccia un cerchietto nero pieno, con ascissa del centro L*80+N*25, ordinata Y del centro 75-(N*5) e raggio di 10. Non ho usato nessuna formula speciale per questi numeri; come con la mag-

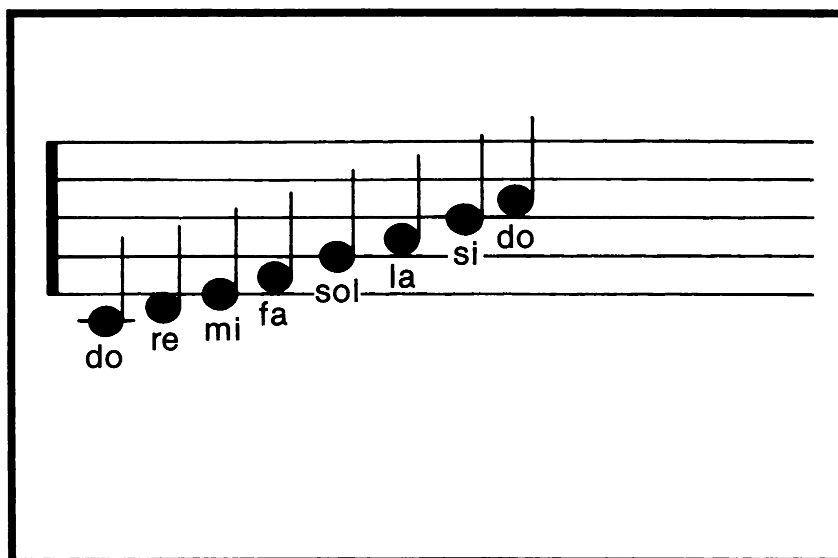


Figura 5.3: pentagramma musicale e posizione delle note.

Durata	Tipo di nota
96	Semibreve (vale quattro quarti)
72	Minima col punto (vale tre quarti)
48	Minima (vale due quarti)
36	Semiminima col punto (vale un quarto e mezzo)
24	Semiminima (vale un quarto)
12	Croma (vale un ottavo)
6	Semicroma (vale un sedicesimo)

Tabella 5.4: durata delle note.

gior parte dei problemi di programmazione, ho iniziato solo con un'idea generale di ciò che volevo ed ho continuato per tentativi successivi finché ho raggiunto il mio scopo.

La gambetta della nota tracciata dal bordo destro del cerchietto verso l'alto di 20 pixel completa la nota musicale. Dato che il raggio di PCIRCLE è 6 di 10 pixel, mi è bastato aggiungere 10 alle due ascisse per avere l'appropriata posizione in senso orizzontale ed ho incrementato l'ordinata del secondo punto di 20 pixel rispetto al primo. L'ordinata del primo resta la stessa del centro (vedi linea 90).

L'ultima parte del programma emette il suono (linea 100), chiude i secondi due loop e interrompe il suono quando il programma è completamente eseguito (linea 110). Contiene anche i dati per i loop, le note e le ottave (linee 120-150).

Come eseguire delle canzoni

Programmare delle canzoni con l'ST non è difficile, poiché potete battere i numeri delle note e delle ottave direttamente. Se non avete familiarità con le note musicali, la Figura 5.3 vi mostra la posizione delle note ovvero quali note corrispondono alle diverse posizioni. Osservate attentamente se la nota è posta direttamente su una linea, se si trova nello spazio tra due linee o se

addirittura (come nel caso della nota Do) al di sotto del pentagramma, staccata da qualsiasi linea.

Il diverso aspetto delle note (breve, semibreve, minima, semiminima, croma eccetera) ne dichiara la durata (che è il quinto argomento dell'istruzione SOUND). La Tabella 5.4 vi dà dei valori della durata stimati con buona approssimazione e la Figura 5.4 vi mostra l'aspetto delle note.

Se volete un programma musicale con il quale dobbiate solo battere i valori nelle istruzioni DATA e far suonare la vostra musica al computer, potete usare il prossimo programma; non è molto elegante, ma funziona.

```
 5 REM Programma musicale generico
10 READ X: REM Legge il numero delle note da suonare
20 FOR N=1 TO X
30 READ NO,OT,DU: REM Legge la nota, l'ottava e la durata
40 SOUND 1,15,NO,OT,DU
50 SOUND 1,0,4,4,0
60 NEXT
70 DATA: REM Metti qui i numeri delle note da suonare
80 DATA: REM metti i numeri delle note, ottave e durate qui
   e su linee successive
```

Ricordate di mantenere l'ordine dei numeri corrispondenti alle note, alle

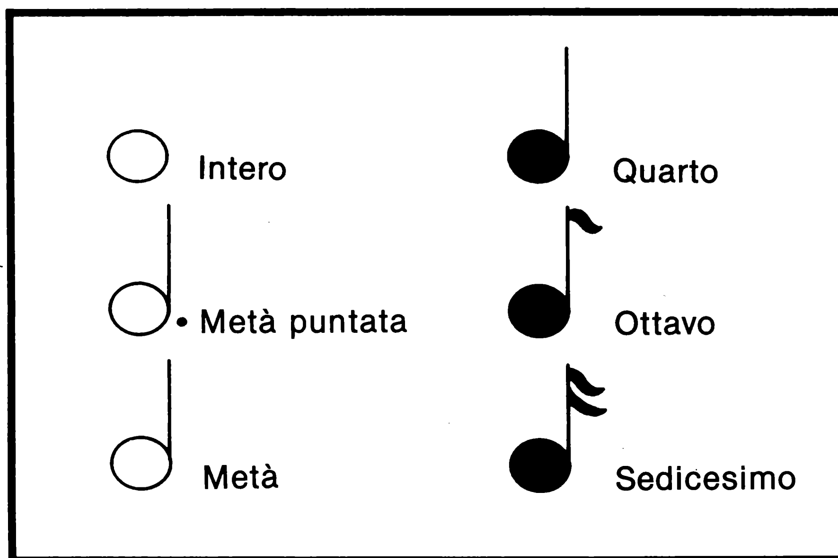


Figura 5.4: tipi di note.

```
5 REM Programma musicista
10 CLEARW 2 : FULLW 2
20 DIM A$(13),N(120),O(120),L(120): X=1
30 INPUT "Inserire il volume (1-15): ";VO: IF VO<1 OR VO>15 THEN GOTO 30
40 FOR I=1 TO 12: READ A$(I)
50 NEXT
60 INPUT "Nota, durata e ottava: ";N$,DU,OT
70 IF N$="QUIT" OR N$="END" THEN GOTO 170
80 GOSUB 120
90 X=X+1
100 GOTO 60
110 DATA DO,DO#,RE,RE#,MI,FA,FA#,SOL,SOL#,LA,LA#,SI
120 NS=1
130 IF A$(NS)=N$ THEN GOTO 150
140 NS=NS+1: GOTO 130
150 N(X)=NS: L(X)=96/DU: O(X)=OT
160 RETURN
170 REM Esecuzione della musica
180 FOR I=1 TO X-1
190 SOUND 1,VO,N(I),O(I),L(I): SOUND 1,0,4,4,0
200 NEXT
210 FOR I=1 TO X-1: PRINT "Nota "N(I)", durata "L(I)", ottava "O(I)
220 NEXT
230 INPUT X
```

Listato 5.2: Programma musicista.

ottave e alla durata, o ascolterete una catastrofe, invece di una sinfonia.

Se non volete usare le tabelle per tradurre la notazione musicale in numeri, il programma del Listato 5.2 vi può aiutare. Accetta informazioni dalla tastiera sulle note che volete suonare, una volta che le avrete inserite suona la canzone e poi visualizza i numeri che dovrete usare per programmare quella canzone.

Il computer inizialmente cancella lo schermo (linea 10), imposta la matrice delle variabili (linea 20) ed accetta il valore relativo al volume (linea 30).

Il programma poi legge in memoria le 12 stringhe delle diverse note (linea 40); i numeri della matrice corrispondono opportunamente ai numeri delle note. Quindi la stringa nella matrice A$(5) corrisponde a quello che la nota 5 rappresenta: Mi. Poi il programma inizia ad accettare dalla tastiera i dati relativi alle note, ottave e durata (linea 60); quando avrete finito d'inserire i dati, basta battere *QUIT,0,0,* oppure *END,0,0* (poiché l'istruzione INPUT prevede di ricevere una nota e due valori numerici).

Nella linea 80, il computer salta alla subroutine che inizia alla linea 120; potete battere queste linee anche senza seguirne l'ordine, dato che le riordina il computer man mano che programmate. La subroutine delle linee 120-160 trova il numero della nota nella stringa che avete appena inserito. Scorre tutte le 12 stringhe contenute in A$(NS) e, quando incontra quella che corrispon-

de, assegna quel numero di nota alla matrice N(X). Per la durata dovete inserire un numero corripondente al valore della nota che volete suonare; alla semibreve (che vale un intero) corrisponde 1, alla minima (che vale due quarti) corrisponde il numero 2, alla semiminima (che vale un quarto) corrisponde 4, alla croma (che vale un ottavo) corrisponde 8 e così via.

Il programma determina la durata delle note dividendo 96 per il numero che voi avete appena inserito ed il numero dell'ottava viene assegnato direttamente ad una matrice.

Nella linea 90, per ciascuna nota che battete, la variabile X viene aumentata di 1 (ricordate che X era fissata a 1 all'inizio del programma nella linea 20); la linea 100 dice al computer di ritornare alla linea 60 per continuare ad accettare l'input e la linea 110 contiene i dati di stringa memorizzati nella matrice A$().

Infine il computer esegue la musica e visualizza i dati numerici che vi servono. La routine che esegue la musica conta semplicemente da 1 a X-1 in modo tale che tutte le note vengano suonate (linea 180) e la seconda istruzione SOUND nella linea 190 interpone una brevissima pausa tra una nota e l'altra. La linea 210 visualizza i dati relativi a note, durata e ottava in modo che li possiate scrivere ed usare direttamente (per esempio nel generico programma musicale che abbiamo visto poco fa).

La linea 230 contiene un'istruzione INPUT il cui unico scopo è di far attendere il computer finché non avrete finito di scrivere i dati; quando avrete finito, premete Return per permettere al computer di sottrarsi all'istruzione INPUT e di finire il programma.

Quando lancerete il programma Musicista (Listato 5.2), dovreste prestare grande attenzione ai dati che inserite per le note e le ottave. La nota deve corrispondere alla stringa che presenta il numero della nota. Se volete che il computer suoni La, inserite *LA*; per Re-diesis inserite *RE#*.

L'ottava deve essere compresa tra 1 e 8; la maggior parte delle canzoni sono nelle ottave centrali. Il computer convertirà tutte queste informazioni in modo che i dati che visualizzerà alla fine del programma saranno già tradotti nei numeri che dovete usare nelle istruzioni DATA.

Vediamo come si presenta l'output del Listato 5.2:

```
RUN
Inserire il volume (1-15): 14
Nota, durata e ottava: DO,2,3
Nota, durata e ottava: MI,4,4
Nota, durata e ottava: SOL,8,5
Nota, durata e ottava: LA#,1,2
Nota, durata e ottava: END,0,0
```

```
5 REM Programma Bacchette.
10 WAVE 0
20 SOUND 1,0,1,1,0: SOUND 2,0,1,1,0
30 FOR X=1 TO 3
40 READ N,N1,N2
50 FOR I=1 TO N
60 SOUND 1,15,N1,4,1
70 SOUND 2,15,N2,4,1
80 WAVE 7: FOR D=1 TO 90: NEXT: WAVE 0
90 NEXT I
100 NEXT X
110 SOUND 1,15,1,4,1
120 SOUND 2,15,1,5,10: WAVE 7: FOR D=1 TO 1250: NEXT: WAVE 0: END
130 DATA 6,6,8
140 DATA 6,5,8
150 DATA 6,3,12
```

Listato 5.3: programma Bacchette.

Nota: la canzone viene eseguita in questo punto.

Nota 1, durata 48, ottava 3
Nota 5, durata 24, ottava 4
Nota 8, durata 12, ottava 5
Nota 11, durata 96, ottava 2

Solo per divertimento, il Listato 5.3 è una canzone già fatta (se una canzone può intitolarsi "Bacchette") che usa due voci contemporaneamente. Prima il programma disinserisce il suono con l'istruzione WAVE (linea 10) e dà due istruzioni SOUND che fissano il volume della prima e della seconda voce a 0 (linea 20).

Il programma ora legge tre set di dati (linea 30), ciascuno dei quali contiene due note (N1 e N2) ed il numero di volte che ciascuna coppia di note deve essere suonata (N). Dopo che ciascun set viene letto, il computer inizia un loop FOR...NEXT per suonare le note, lascia un breve intervallo, poi continua il loop (linee 50-100).

L'ultima parte di questo programma finisce con un'altra coppia di note che vengono suonate solo per un paio d'istanti (linee 110-120). Le istruzioni DATA nelle linee 130-150 contengono i tre set di note che servono per suonare la maggior parte di "Bacchette".

Suoni con SOUND

Usando una durata 0, potete trasformare l'istruzione SOUND da musicista a rumorista. Anche se continuerete ad usare note ed ottave, potete mescolarle ad alta velocità ed ottenere suoni interessanti per videogiochi, programmi educativi o altre applicazioni. Per fare effetti sonori con l'istruzione SOUND userete quasi sempre un loop FOR...NEXT oppure dei modelli casuali.

Il primo esempio di programma impiega entrambe le tecniche per creare una cacofonia di suoni. Tutti i canali sono attivati con l'istruzione WAVE 15 e ciascuna voce produce un suono a caso. Il loop FOR...NEXT porta il computer sulle tre voci e il GOTO 20 della linea 50 forma un loop infinito in modo che il suono continui finché non fermerete il programma con Control-C.

```
 5 REM Cacofonia di suoni
10 WAVE 15
20 FOR V=1 TO 3
30 SOUND V,RND*15,RND*12,RND*8,0
40 NEXT
50 GOTO 20
```

Ricorrere ai loop è un buon modo di produrre suoni, poiché la maggior parte dei suoni caratteristici implicano variazioni continue di frequenza. Vediamo ora un programma che scorre l'intera gamma delle 96 note ad alta velocità cosicché dal primo canale viene emesso un suono rapidamente ascendente.

```
 5 REM Tono ascendente
10 WAVE 1
20 FOR I=1 TO 8: REM Si sposta lungo otto ottave
30 FOR J=1 TO 12: REM e 12 note
40 SOUND 1,15,J,I,0
50 NEXTJ: NEXT I
```

Usando tutte e tre le voci per un effetto simile, si produce con il prossimo programma una serie di toni ancora più interessante. La voce 1 va dall'ottava 1 alla 6, la voce 2 va dall'ottava 2 alla 7 mentre la voce 3 va dall'ottava 3 alla 8. Tutti i canali sono usati contemporaneamente per emettere 56 diverse note e siccome ogni canale è in un'ottava diversa, le tre note insieme formano un effetto speciale.

Nelle linee 20-40 le tre voci vengono attivate e hanno inizio i loop per l'ottava (il loop I), per la nota (il loop J) e le tre voci (il loop K).

```
 5 REM Tre voci
10 WAVE 15
20 FOR I=0 TO 5
30 FOR J=1 TO 12
40 FOR K=1 TO 3
```

Poi si usa SOUND insieme alle variabili FOR...NEXT per mandare ogni nota al canale appropriato. Le variabili I e K sono state aggiunte per determinare l'ottava e poiché K rappresenta la voce usata al momento, questa tecnica assicura che ciascun canale userà un'ottava diversa (uguale al numero del canale più il numero dell'ottava in questione). La linea 60 chiude i tre loop:

```
50 SOUND K,15,J,I+K,0
60 NEXT K: NEXT J: NEXT I
```

Produrre suoni diversi da due o tre canali è interessante quando si tratta della stessa nota, ma potete ottenere effetti sonori particolari, usando note dissonanti e cambiamenti di frequenza nei diversi canali. In questo prossimo programma, il primo canale emette un suono rapidamente ascendente, mentre il secondo emette un suono rapidamente discendente. Le frequenze si avvicinano, diventano uguali e poi si differenziano di nuovo finché un suono finisce dove l'altro era iniziato. Ciò è possibile semplicemente ricorrendo ai valori I (per l'ottava) e J (per la nota) nell'istruzione SOUND della linea 40, poi invertendo i valori specificando 9—I (per avere l'accordo opposto) e 13—J (per avere la nota opposta) nell'istruzione SOUND della linea 50. Usando le variabili in loop FOR...NEXT in modi insoliti come questo si semplifica la programmazione di effetti sonori complessi.

```
 5 REM Cambiamenti di frequenza
10 WAVE 7
20 FOR I=1 TO 8
30 FOR J=1 TO 12
40 SOUND 1,15,J,I,0
50 SOUND 2,15,13—J,9—I,0
60 NEXT J: NEXT I
70 WAVE 0
```

Come visualizzare i suoni

Quando la frequenza di un suono aumenta e diminuisce, potrebbe essere interessante vedere il grafico del suono mentre viene emesso. Questi grafici

non sono difficili da tracciare, dato che basta associare il valore dell'ordinata alla frequenza del suono; toni più alti faranno salire la linea del grafico verso l'alto, mentre toni più bassi porteranno la linea verso il basso.

Diversamente dagli altri programmi di questo libro, i prossimi due vanno svolti nella modalità di risoluzione media, per poter visualizzare nel grafico il maggior numero di suoni possibile; se vi trovate nella modalità di bassa risoluzione, tornate alla scrivania GEM e usate Set Preferences per passare alla risoluzione media per i prossimi due programmi.

Il primo programma che combina grafica e suoni stampa il grafico di suoni casuali mentre li emette. Il programma cancella lo schermo (linea 10), attiva il primo canale e traccia le linee orizzontali del reticolo usando un loop FOR...NEXT:

```
 5 REM Stampa il grafico di suoni casuali
 7 REM Usa la risoluzione media
10 FULLW 2: CLEARW 2:WAVE 1
20 COLOR 1,1,1: FOR X=10 TO 590 STEP 20
30 LINEF X,10,X,160: NEXT X
```

Poi si tracciano le linee vericali con un secondo loop e si cambia il colore in modo che il grafico risalti sul reticolo sottostante.:

```
40 FOR Y=10 TO 160 STEP 15
50 LINEF 10,Y,590,Y: NEXT Y: COLOR 1,1,3
```

Per far scorrere il grafico lungo lo schermo, il loop conta da 10 a 585 in incrementi di 5; il grafico si disegna collegando i punti con segmenti. Il programma ricorda le vecchie ordinate registrandole nella variabile OYC e l'ordinata nuova è stabilita a caso e registrata in YC. Dato che all'inizio del programma non c'è ancora un'ordinata, OYC viene fissato arbitrariamente a 100 quando il loop comincia:

```
60 OYC=100: FOR X=10 TO 585 STEP 5
```

Ora vengono selezionate note ed ottave casuali (linea 70) e dopo che ogni nota viene emessa (linea 80), viene scelta una nuova ordinata e viene tracciato un segmento che unisce la vecchia ordinata alla nuova. I due valori dell'ascissa sono rispettivamente i valori correnti di X dal loop FOR...NEXT e quello stesso valore più 5; YC è memorizzato in OYC prima che il loop continui alla linea 100 e quando il loop è finito, WAVE 0 nella linea 110 interrompe il suono:

```
 70 O=RND*8: N=RND*12
 80 SOUND 1,15,N,O,3
 90 YC=160-(O*12+N): LINEF X,OYC,X+5,YC: OYC=YC
100 NEXT X
110 WAVE 0
```

Cambiando le semplici variabili in matrici, potete creare un grafico simile che usa tre diversi canali, ciascuno dei quali emette suoni casuali. Il grafico è quello della Figura 5.5 ed il programma è quello del Listato 5.4.

Notate in particolare che nella linea 80 l'istruzione COLOR è usata associata al valore di A in modo che i tre diversi grafici si presentino in tre diversi colori. Nella modalità di risoluzone media, sono disponibili solo quattro colori di cui uno è il colore dello sfondo. Tuttavia, usando tre colori diversi da quello dello sfondo per i tre grafici risulta comunque più facile distinguerli.

Effetti sonori

Ora vediamo quali effetti potete produrre variando gli argomenti dell'istruzione WAVE. Per darvi un'idea del significato degli argomenti di WAVE, il prossimo programma impiega tutte le forme d'onda e una vasta gamma di

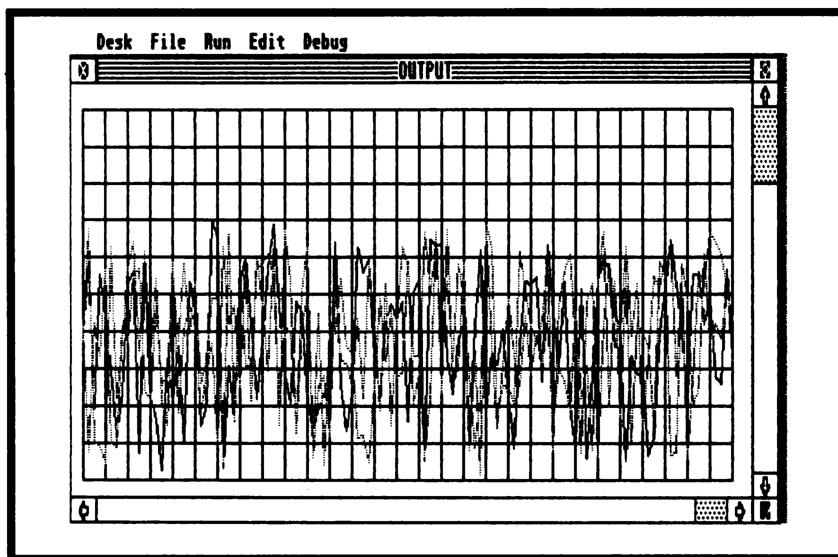


Figura 5.5: tre voci casuali, visualizzate mentre vengono emesse.

```
5 REM Voci casuali, visualizzate mentre suonano.
7 REM Impiega la risoluzione media
10 FULLW 2: CLEARW 2: WAVE 15: REM tutte le tre voci vengono attivate
20 COLOR 1,1,1: FOR X=10 TO 590 STEP 20
30 LINEF X,10,X,160: NEXT X
40 FOR Y=10 TO 160 STEP 15
50 LINEF 10,Y,590,Y: NEXT Y
60 OYC(1)=100: OYC(2)=100: OYC(3)=100: FOR X=10 TO 585 STEP 5
70 FOR A=1 TO 3: O(A)=RND*8: N(A)=RND*12
80 SOUND 1,15,N(A),O(A),1: COLOR 1,1,A
90 YC(A)=160-(O(A)*12+N(A)): LINEF X,OYC(A),X+5,YC(A): OYC(A)=YC (A)
100 NEXT A: NEXT X
110 WAVE 0
```

Listato 5.4: programma di voci casuali.

periodi, poi emette i suoni. In effetti la finalità dell'argomento periodo è più ampia in modo che non dovete solo usare i numeri da 1 a 31 per modulare i bassi e gli alti; usando numeri molto grandi (per esempio 1000) per il periodo, potete aumentare la gamma dei suoni che il ST può emettere. Notate come cambia il tono quando cambiano gli argomenti di WAVE:

```
 5 REM Ciclo tra forme d'onda e periodi
10 FULLW 2: CLEARW 2
20 FOR X=1 TO 8: READ ENV
30 PRINT "Questa è la forma";ENV
40 FOR D=1 TO 500: WAVE 8,3,ENV,D
50 FOR D1=1 TO 300: NEXT D1
60 NEXT D: NEXT X
70 DATA 0,4,8,10,11,12,13,14
80 WAVE 0
```

Il prossimo programma visualizza il numero della forma e del periodo mentre emette il suono, sovrapponendo i nuovi valori su quelli del suono precedente, ed adotta una gamma molto più ristretta per il periodo:

```
 5 REM Visualizza i numeri di forma e il periodo mentre emette i suoni
10 FULLW 2: CLEARW 2
```

```
20 FOR SHAPE=8 TO 14
30 FOR PERIOD =0 TO 31
40 WAVE 1,1,SHAPE,PERIODO,3
50 GOTOXY 0,0: PRINT "Forma=" SHAPE", periodo=" PERIOD
60 NEXT
70 NEXT
RUN
```

Forma = 8, periodo = 0
Forma = 8, periodo = 1
Forma = 8, periodo = 2
.
.
.
Forma = 14, periodo = 31

Oltre a poter alterare la natura del suono, WAVE può produrre effetti sonori in modo completamente autonomo. Purtroppo non esiste un modo sistematico di produrre proprio il suono che desiderate. Come sempre, provare con i numeri casuali vi aiuterà a imbattervi in diversi effetti interessanti e troverete in fretta quello che cercate.

Concentratevi sul cambiamento del valore del periodo, dato che valori alti in questo argomento danno buoni risultati. Mentre provavo diversi valori con WAVE, ho incontrato diversi effetti sonori interessanti, che ho raccolto nel prossimo programma:

```
  5 REM Effetti sonori
 10 FULLW 2: CLEARW 2
 20 PRINT "Onde sulla spiaggia"
 30 WAVE 8,3,14,9000
 40 GOSUB DELAY
 50 PRINT "Elicottero:"
 60 WAVE 8,3,14,900
 70 GOSUB DELAY
 80 PRINT "Batteria sintetizzata:"
 90 SOUND 1,15,1,4,1: SOUND 2,15,5,4,1: SOUND 3,15,8,4,1
100 WAVE 56,7,8,1500
110 GOSUB DELAY
120 SOUND 1,0: SOUND 2,0: SOUND 3,0
130 END
140 DELAY: FOR D=1 TO 9000: NEXT: RETURN
```

I SUONI ED IL MOUSE

Così come si può combinare la grafica con i suoni, potete associare altri componenti dell'ST. Vediamo qui come potete usare il mouse per modificare il suono che viene emesso. Il computer può ricevere informazioni direttamente dal mouse ed usando nelle istruzioni acustiche i valori che riceve, vi permette di controllare i suoni prodotti. Con il prossimo programma, potete spostare il mouse su e giù per cambiare gli accordi e da destra a sinistra per cambiare le note. Iniziate aprendo il primo canale, cancellando lo schermo e impostando una routine per l'input dal mouse (vedi il paragrafo sulle operazioni GEMSYS nel Capitolo 6 per i dettagli):

```
  5 REM Usa il mouse per modificare il suono
 10 WAVE 1
 20 FULLW 2: CLEARW 2
 30 A# =GB: CO=1
 40 CONTROL=PEEK(A#)
 50 GLOBAL=PEEK(A# +4)
 60 GINTIN=PEEK(A# +8)
 70 GINTOUT=PEEK(A# +12)
 80 ADDRIN=PEEK(A# +16)
 90 ADDROUT=PEEK(A# +20)
100 GEMSYS (79)
```

Il computer riceve dal mouse le ascisse e le ordinate e le raccoglie nelle variabili X e Y rispettivamente. Questi valori possono poi essere usati come argomenti delle note e delle ottave nell'istruzione SOUND:

```
110 X=PEEK (GINTOUT+2): Y=PEEK (GINTOUT+4)
120 SOUND 1,15,INT (X/56)+1,INT (Y/27)+1,0: GOTO 100
```

Il programma seguente, che è simile, vi permette di spostare il mouse per cambiare il periodo e di premere il tasto Alternate per avere forme diverse.

```
 5 REM Usa il mouse per cambiare periodo
10 WAVE 1: SHAPE=8
20 FULLW 2: CLEARW 2
30 A# =GB: CO=1
40 CONTROL=PEEK(A#)
50 GLOBAL=PEEK(A# +4)
60 GINTIN=PEEK(A# +8)
```

```
 70 GINTOUT = PEEK(A# + 12)
 80 ADDRIN = PEEK(A# + 16)
 90 ADDROUT = PEEK(A# + 20)
100 GEMSYS (79)
110 GOTOXY 1,1,: PRINT "Periodo = "X + Y",e forma = ";SHAPE
120 X = PEEK (GINTOUT + 2): Y = PEEK (GINTOUT + 4)
130 IF PEEK (GINTOUT + 8) = 8 THEN SHAPE = SHAPE + 1:
    IF SHAPE > 14 THEN SHAPE = 8
140 WAVE 8,1,SHAPE,X + Y: GOTO 100
```

POSSIBILI USI DEI SUONI

Molte persone credono che le capacità acustiche di un computer servano solo nei videogiochi. Sarebbe difficile sostenere che i videogiochi non sono il campo di maggiore applicazione dei suoni, ma ci sono molte altre possibili applicazioni. Potreste includere dei suoni in un programma serio: certi suoni possono indicare all'utente d'inserire i dati, mentre altri potrebbero segnalare gli errori. I programmi educativi potrebbero impiegare un motivetto piacevole quando lo studente risolve brillantemente un problema e per i più piccoli gli effetti sonori e la musica possono aiutare a tenerli concentrati su quello che compare sullo schermo.

Infine potete usare l'ST come un serio strumento musicale, se siete interessati a computerizzare uno spartito musicale o se volete scrivere una canzone voi stessi con l'aiuto del vostro ST. La porta MIDI incorporata rende facile il collegamente del vostro ST ad un sintetizzatore. Accessibile per il costo e molto popolare è il Casio CZ-101, che costa intorno alle 500 mila lire.

Potrete provare uno dei seguenti progetti o realizzarne di vostri:

- Provate a costituire una biblioteca di suoni provando diversi valori nelle istruzioni WAVE e SOUND. Tutte le volte che ottenete qualcosa d'interessante, trascrivete tutti i numeri necessari. Potete facilitarvi il compito scrivendo un programma che vi aiuti a trovare i suoni.

- Scrivete un programma per disegnare i suoni che vi permetta di usare il mouse per tracciare il grafico dei suoni. Quando il grafico è finito, il computer dovrebbe essere in grado di eseguire i suoni alla velocità che desiderate.

- Programmate un'utility acustica per vi permetta di specificare busta, periodo, nota, ottava e gli altri valori per produrre il suono che descrivete; magari potreste aggiungervi la grafica; per esempio, quando il programma chiede

la forma, il computer potrebbe visualizzare tutte le possibili forme d'onda insieme ai relativi numeri.

- Traducete lo spartito di un pezzo musicale che vi piace in un programma che lo esegua. Potrete anche usare due o tre voci per un pezzo più complicato, anche se l'intento può nascondere qualche insidia. Provate ad usare l'istruzione WAVE per dare al suono le caratteristiche appropriate, in base al tipo di musica che programmate.

Anche se l'ST dispone solo di due comandi per produrre suoni e musica, sono entrambi versatili e potenti. Ciò vi facilita la programmazione. Una volta che avrete padronanza anche sui dettagli di questi due comandi ed imparato come usarli in modo efficiente, troverete che programmare è ancora più divertente e che anche la qualità delle vostre prestazioni ne trarrà vantaggio.



GRAFICA AVANZATA E PROGRAMMAZIONE DI TESTO

Due parole chiave del BASIC ST - VDISYS e GEMSYS - vi danno accesso ad una tale verietà di funzioni da costituire quasi un linguaggio a sé stante. In questo capitolo studierete nei dettagli uno di questi linguaggi, le funzioni dell'Interfaccia di Dispositivo Virtuale del GEM, abbreviato in VDI, che possono essere incorporate nei vostri programmi BASIC con l'istruzione VDISYS. Molte di questo funzioni sono più veloci e più facili da usare dei loro corrispondenti in BASIC; altre non hanno addirittura alcun equivalente diretto.

GEMSYS, che dà accesso all'Application Environment System, o AES è molto meno rilevante per chi programma in BASIC e ne parleremo brevemente solo alla fine di questo capitolo. Per quanto VDISYS sia una delle istruzioni più potenti del BASIC ST, il VDI viene quasi completamente trascurato nella documentazione dell'Atari. Vi sarà utile avere una qualche conoscenza di base del sistema prima d'impararne le singole funzioni.

L'ISTRUZIONE VDISYS

Alcune delle funzioni che saranno descritte in questo capitolo potrebbero sembrarvi prive d'importanza, dato che possono essere svolte (forse più facilmente) in BASIC. Per esempio, probabilmente trovereste più semplice usare l'istruzione CIRCLE piuttosto che usare tre o quattro linee per disegnare un cerchio usando la funzione VDI equivalente. Tuttavia VDI e AES offrono i seguenti vantaggi:

- Una volta che avete inserito tutti i parametri in memoria (nel caso di un cerchio il centro, il colore e il raggio), potete cambiare anche una sola caratteristica ed eseguire la routine ripetutamente per ottenere un effetto diverso. Potreste, per esempio, cambiare il raggio e disegnare un cerchio più grande o più piccolo del precedente senza dover ripetere anche gli altri argomenti.

- Molte delle funzioni del VDI non hanno un equivalente in BASIC. Dato che queste funzioni sono incorporate nella ROM, disporrete di un'intera biblioteca di strumenti veloci e facili da usare che non sono altrimenti accessibili nel BASIC ST.

- La vostra comprensione del computer sarà decisamente migliore quando avrete usato per un pò le routine incorporate. La comprensione di POKE, byte, indirizzi e di altri elementi di questi codici di controllo vi daranno una base più consistente per capire come l'ST funziona internamente. Ciò vi faciliterà anche l'apprendimento di altri linguaggi e di tecniche più avanzate.

Le routine VDI impiegano diversi nomi di variabile che contengono specifici indirizzi di memoria che l'ST può usare. Non è necessario che fissiate alcun valore per queste variabili, dato che il computer conserva sempre in memoria il loro valore corretto. Le variabili che stabiliscono i parametri VDI sono

CONTRL (stabilisce la matrice di controllo in uso)
INTIN (numero dei parametri d'input)
PTSIN (coordinate del punto d'input)
INTOUT (parametri di output)
PTSOUT (coordinate del punto di output)

Impiegando questi nomi di variabile (ed incrementandoli in espressione del tipo INTIN+2 o CONTRL+6), potrete usare POKE in certe celle di memoria per stabilire i parametri di una chiamata VDI senza dovervi preoccupare di determinare gli indirizzi di memoria.

Input di VDISYS

POKE

La cosa più importante che dovete sapere quando impiegate l'istruzione VDISYS è il significato della parola chiave POKE del BASIC ST. Ho usato brevemente questa istruzione nel Capitolo 4, per inserire una nuova tavolozza di colori nella memoria del computer. In generale, la memoria di un computer è formata da *celle* contigue, misurate in byte. Con POKE potete inserire un valore (da 0 a 255) in una cella (il formato di POKE è già stato dato nel Capitolo 4). Per mettere il valore 100 nella cella 230, basta battere *POKE 230,100.* POKE è utile poiché dà al computer un modo veloce e regolare di recuperare le informazioni; una specifica area di memoria, ad esempio, può contenere il valore del colore dello schermo ed il computer può individuare quel byte per eventuali cambiamenti di colore apportati dal programmatore.

POKE è importante quando si usa VDISYS perché si devono solitamente specificare da cinque a sette byte di memoria per dire al computer che cosa esattamente volete fare. Nel BASIC ST, usate gli argomenti per far fare qualcosa al computer; per esempio, si può disegnare un cerchio con il centro in (50,60) e con il raggio di 35 battendo:

```
CIRCLE 50,60,35
```

Con le istruzioni di sistema, invece, si usa POKE per inserire i diversi valori

(ed alcuni valori che non cambiano mai per la funzione) in specifici byte di memoria. Fare ciò è molto facile, poiché quando scrivete il programma che impiega la funzione dovete solo leggere le informazioni sulla parola chiave e seguire un modello prestabilito.

Standard di VDISYS

I parametri d'input per l'istruzione VDISYS sono relativamente numerosi; anche se le diverse routine hanno numeri diversi per le varie celle di memoria, vediamo un elenco delle più probabili finalità di alcune celle di memoria:

CONTRL (0) Il codice delle funzione VDISYS che state per usare è registrato in CONTRL (0). Per esempio, il codice di controllo per stabilire il colore di una forma piena è 25, quindi CONTRL (0)=25 dice al computer di usare quella routine.

CONTRL (5) Il numero della primitiva grafica generalizzata (GDP ovvero Generalized Drawing Primitive) è contenuto nella cella CONTRL (5). Le funzioni GDP vengono impiegate per mandare la grafica allo schermo e i diversi numeri memorizzati con *POKE CONTRL+10* diranno al computer quale funzione grafica usare.

PTSIN Il computer cerca gli specifici argomenti per una funzione grafica nell'area determinata dal valore PTSIN. Se disegnate una linea sullo schermo, potete inserire le due coppie di coordinate in PTSIN, PTSIN+2, PTSIN+4 e PTSIN+6.

La maggior parte delle routine VDISYS richiedono più istruzioni POKE di quelle elencate prima. Ricordate anche che i valori di PTSIN, INTIN, CONTRL e altri nomi di variabili usati direttamente con VDISYS sono delle costanti; gli Atari ST hanno questi valori incorporati per facilitarvi la programmazione. Usare *POKE CONTRL,2* è più facile che cercare di calcolare con precisione in che punto della memoria dovete mettere un certo numero.

Un esempio di VDISYS

Per esempio, queste sono le informazioni che vi servono per il Codice di Controllo 11, che traccia un cerchio sullo schermo:

CONTRL (0)=11
CONTRL (1)=3

CONTRL (3)=0
CONTRL (5)=4
PTSIN (0)= X-ascissa del centro
PTSIN (1)= Y-ordinata del centro
PTSIN (4)= raggio

Ora che avete queste informazioni, dovrete solo battere le istruzioni POKE necessarie per ottenere il cerchio che desiderate. Vediamo il formato di ciascun componente:

POKE *variabile VDI+numero*2,valore*

La *variabile VDI* può essere CONTRL, PTSIN o INTIN; il *numero* è il numero tra parentesi che accompagna la variabile VDI - è importante moltiplicarlo per due, perché gli inserimenti vengono memorizzati in parole (16 bit) e non in byte; il *valore* è il numero che volete mettere in quella cella di memoria. Confrontate le informazioni originali con quello che dovete battere voi per vedere quale relazione intercorre:

```
CONTRL (0)=11
POKE CONTRL+0,11
```

In effetti non c'è bisogno di battere *+0* quando il valore del numero tra parentesi è 0. Potreste battere *POKE CONTRL,11* per avere lo stesso risultato.

Quello che segue è un programma completo per disegnare un cerchio con il centro in (100,30) e raggio di 20; notate che le istruzioni REM contengono le informazioni d'input che ho usato per ottenere le istruzioni POKE:

```
 4 REM Cerchio
 6 DEF SEG = 0
 8 COLOR 1,1,1,1,1: FULLW 2
10 POKE CONTRL,11: REM CONTRL (0)=11
20 POKE CONTRL+2,3: REM CONTRL (1)=3
30 POKE CONTRL+6,0: REM CONTRL (3)=0
40 POKE CONTRL+10,4: REM CONTRL (5)=4
50 POKE PTSIN,100: REM PTSIN (0)= X-ascissa del centro
60 POKE PTSIN+2,30: REM PTSIN (1)= Y-ordinata del centro
70 POKE PTSIN+8,20: REM PTSIN (4)= raggio
80 VDISYS (1)
```

Notate la prima istruzione, *DEF SEG = 0*. L'istruzione DEF SEG, il cui formato è

DEF SEG = *espressione numerica*

si usa associata a POKE (e al relativo PEEK). L'*espressione numerica* dice al computer due cose: il numero di byte che le successive istruzioni POKE e PEEK devono inserire o leggere e qualsiasi equivalente all'indirizzo specificato in quelle istruzioni. Il valore di 0 usato in questa istruzione dice al computer di far operare POKE su due byte per volta e di non usare alcun equivalente.

Notate anche l'ultimo comando, nella linea 80: *VDISYS (1)*. Esso attiva la funzione che avete appena indicato nella memoria. Che abbiate attivato una funzione per ottenere un certo risultato o per esaminare una parte della memoria, ricordate di battere *VDISYS (1)* per lanciare la routine.

Se vi ricordate di usare il nome di variabile corretto (PTSIN, INTIN e CONTRL) e se moltiplicate l'intero tra parentesi per due per determinare quanto va aggiunto al nome della variabile, il vostro programma VDISYS dovrebbe funzionare a dovere.

Output VDISYS

PEEK

Un'ultima cosa da sapere è che talvolta sarete in grado di ottenere importanti informazioni dalla funzione VDISYS del computer esaminando il contenuto di una cella di memoria anziché inserendo un valore in una di esse. Potete far questo usando l'istruzione PEEK, la quale, come POKE, tratta uno specifico byte in memoria. Il formato di PEEK è:

PEEK *(indirizzo)*

dove *indirizzo* è la cella di memoria del byte che volete esaminare. PRINT PEEK (400), quindi, vi dirà che cosa conteneva il valore alla cella 400 nella memoria del computer (secondo l'istruzione DEF SEG precedente).

Con le routine VDISYS, PEEK è spesso utile per trovare informazioni come le coordinate del cursore sullo schermo, il colore in uso, la forma che sta per essere tracciata e così via. Se state usando una particolare funzione VDISYS, ed avete battuto *VDISYS (1)* per attivarla, potete talvolta controllare certe parti della memoria per raccogliere informazioni. Ogni volta che vedrete una sezione Output elencata in una qualsiasi descrizione del Codice di Controllo, usate PEEK per esaminare la parola elencata semplicemente usando il nome della variabile ed aggiungendo il numero tra parentesi (anche qui, moltiplicato per due):

PEEK *(variabile VDI+numero*2)*

Non dimenticate di battere prima *DEF SEG = 0.* Per esempio, supponete di voler esaminare PTSOUT (4). Dato che 4 per 2 fa 8, dovrete battere:

```
PRINT PEEK (PTSOUT+8)
```

Se volete mettere il contenuto di INTOUT (5) nella variabile X, batterete:

```
X=PEEK (INTOUT+10)
```

Come con qualsiasi funzione VDISYS, la dovete attivare con VDISYS (1) prima di poter esaminare qualsiasi informazione.

Codici di controllo

Quello che segue è un elenco delle più importanti funzioni GEM a cui potete accedere con l'istruzione VDISYS. Il primo paragrafo riguarda le funzioni orientate alla grafica, mentre il secondo tratta le funzioni di testo. Esse sono in ordine numerico, in modo che possiate usare questo capitolo anche per la consultazione. Prendetevi pure la libertà di esaminarle in un ordine qualsiasi, dato che già possedete le informazioni necessarie per attivare una qualsiasi di queste funzioni.

CODICI DI CONTROLLO PER LE FUNZIONI GRAFICHE VDISYS

Codice di controllo 6: Polyline (Linee multiple)

La funzione *polyline* disegna una figura usando i punti che voi specificate. L'analogia migliore per questa routine è un disegno punto per punto, poiché il computer accetta le informazioni relative a diversi punti sullo schermo e poi traccia delle linee da un punto all'altro, a cominciare dal primo punto che inserite per finire all'ultimo.

È importante ricordare che il computer non chiude automaticamente l'area delimitata in base ai punti da voi specificati. Se date al computer i tre vertici di un triangolo, traccerà i segmenti dal primo punto al secondo al terzo, ma non completerà il triangolo riandando al primo punto: dovrete nuovamente

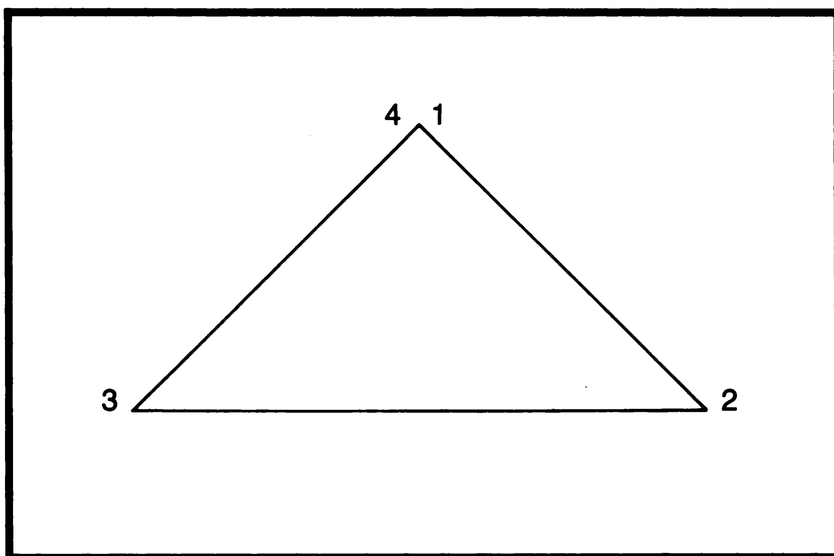


Figura 6.1: servono quattro punti per completare il triangolo.

specificare il primo punto (come quarto punto) perché il computer chiuda il triangolo, come mostra la Figura 6.1.

Celle di memoria

CONTRL (0)=6
CONTRL (1)=n (numero di coppie di coordinate)
CONTRL (3)=0
PTSIN (0)=Prima ascissa
PTSIN (1)=Prima ordinata
PTSIN (2)=Seconda ascissa
PTSIN (3)=Seconda ordinata
.
.
.
PTSIN (n-1)=Ultima ascissa
PTSIN (n)=Ultima ordinata

Esempio di programma. Il programma seguente stampa delle linee colorate per tutto lo schermo. Dopo che il colore della linea viene impostato a caso, il loop FOR...NEXT della linea 30 conta quattro coppie casuali di punti ed

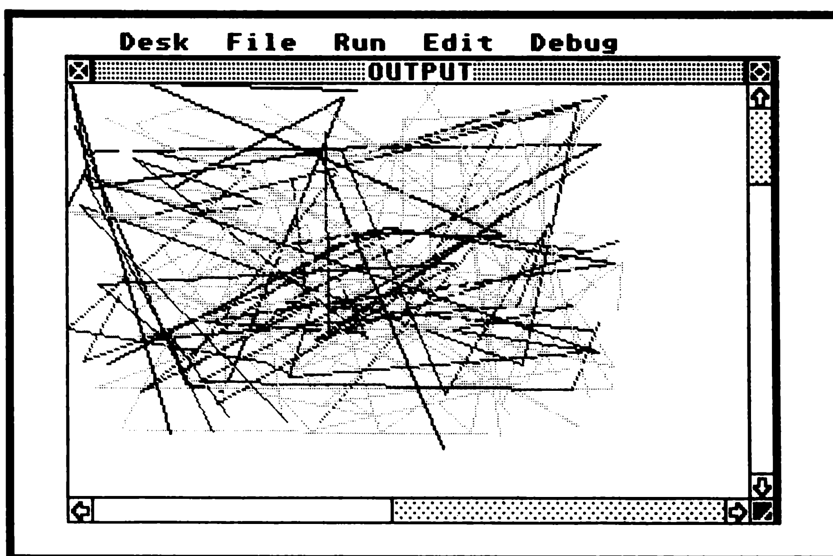


Figura 6.2: riempiamo lo schermo usando la funzione polyline.

impiega POKE 5 per specificarne la posizione. Notate che uso STEP 2 nel loop in modo tale che le coppie di coordinate possano essere messe in memoria con POKE, come PTSIN (I*2) e PTSIN (I*2)+1; l'ascissa va nella prima cella e l'ordinata va nella seconda. Ogni volta che il loop è completo, VDISYS (1) viene attivata per visualizzare le linee, poi un'istruzione GOTO 30 fa ripetere il loop indefinitamente. Poco dopo aver lanciato questo programma, l'intero schermo si riempirà di linee colorate. La Figura 6.2 mostra le linee, ma non i colori.

```
 2 REM Uso della funzione polyline per riempire lo schermo
 4 DEF SEG = 0
 6 FULLW 2: CLEARW 2
10 POKE CONTRL,6
15 POKE CONTRL+2,5
20 POKE CONTRL+6,0
30 COLOR 1,1,RND*15: FOR I=0 TO 8 STEP 2
40 POKE PTSIN+(I*2),RND*250
50 POKE PTSIN+(I+1)*2,RND*150+20
60 NEXT
70 VDISYS (1)
80 GOTO 30
```

Codice di controllo 7: Polymarker (Punti multipli)

Se volete disegnare dei punti sullo schermo, per esempio come per segnare dei punti su una cartina, usate la funzione *polymarker.* Potete specificare tutte le coordinate che volete e, come scoprirete più avanti in questo capitolo, potete cambiare questi punti in altre forme, quali stelle o crocette, con un'altra funzione VDISYS.

Celle di memoria

CONTRL (0)=7
CONTRL (1)=n (numero di coppie di coordinate)
CONTRL (3)=0
PTSIN (0)=Ascissa del primo punto
PTSIN (1)=Ordinata del primo punto
PTSIN (2)=Ascissa del secondo punto
PTSIN (3)=Ordinata del secondo punto
.
.
.
PTSIN (n-1)=Ultima ascissa
PTSIN (n)=Ultima ordinata

Codice di controllo 8: visualizza testo sullo schermo

Anche se potete facilmente stampare il testo sullo schermo con l'istruzione PRINT, potete stampare *testo grafico* usando la funzione VDISYS. Il testo grafico è molto più versatile del testo normale: potete stampare una grande quantità di diversi stili, allineare il testo, ruotarlo in diverse direzioni e svolgere molte altre funzioni per rende la vostra stampa più versatile.

Celle di memoria

CONTRL (0)=8
CONTRL (1)=1
CONTRL (3)=n (numero di caratteri, compresi gli spazi, che volete stampare)
INTIN (0)=Codice ASCII per il primo carattere (vedi Appendice D)

INTIN (1)=Codice ASCII per il secondo carattere

.

.

.

INTIN (n)=Codice ASCII per l'ennesimo carattere
PTSIN (0)=Ascissa del punto dove inizia la stampa
PTSIN (1)=Ordinata del punto dove inizia la stampa

Codice di controllo 9: Filled poligon (Riempimento poligoni)

Simile alla funzione *polyline*, quella di *filled poligon* disegna un poligono di cui sono stati fissati i vertici e quindi lo riempie di colore. Diversamente della funzione *polyline,* questa non richiede, per la chiusura del poligono, di specificare di nuovo il vertice di partenza; la chiusura viene automaticamente eseguita dal computer.

Celle di memoria

CONTRL (0)=9
CONTRL (1)=n (numero di caratteri specifici)
CONTRL (3)=0
PTSIN (0)=Prima coordinata X
PTSIN (1)=Prima coordinata Y
PTSIN (2)=Seconda coordinata X
PTSIN (3)=Seconda coordinata Y

.

.

.

PTSIN (n—1)=Coordinata X
PTSIN (n)=Coordinata Y

Esempio di programma Vediamo ora un programma che disegna poligoni a cinque vertici colorati a caso. Il computer specifica cinque coppie di numeri casuali per individuare i cinque pixel, poi l'istruzione VDISYS fa apparire il poligono sullo schermo. Ritornando alla linea 30, il programma può selezionare un altro colore casuale e visualizzare un altro poligono; ripete questo loop all'infinito. La Figura 6.3 mostra l'output di questo programma.

```
2 REM Poligoni pieni
4 DEF SEG = 0
```

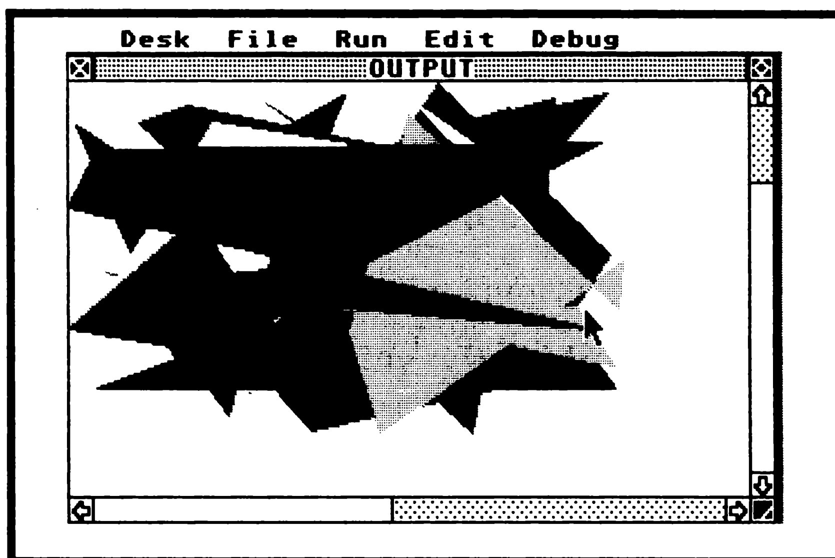


Figura 6.3: poligoni pieni.

```
 6 FULLW 2: CLEARW 2
10 POKE CONTRL,9
15 POKE CONTRL+2,5
20 POKE CONTRL+6,0
30 COLOR 1,RND*15,1: FOR I=0 TO 8 STEP 2
40 POKE PTSIN+(I*2),RND*250
50 POKE PTSIN+(I+1)*2,RND*150+20
60 NEXT
70 VDISYS (1)
80 GOTO 30
```

Codice di controllo 11: grafica primitiva generalizzata (GDP)

Anche se le funzioni GDP sono sotto un unico codice - CONTRL (0)=11 - potete disegnare una grande varietà di forme e di linee modificando i valori registrati in aree specifiche della memoria del computer. Qui di seguito sono elencate tutte le celle di memoria, ma per semplicità selezionate il particolare oggetto che volete tracciare ed usate il codice corrispondente alla descrizione dell'oggetto.

Celle di memoria

CONTRL (0)=11
CONTRL (1)=n (numero di coppie di coordinate usate per disegnare la routine)
CONTRL (3)=0
CONTRL (5)=Numero della forma che volete disegnare:
1 Rettangolo pieno
2 Arco (semicerchio)
3 Settore circolare pieno (come nei diagrammi circolari)
4 Cerchio pieno
5 Ellisse piena
6 Arco ellittico
7 Settore di ellissi piene
8 Rettangolo con angoli arrotondati
9 Rettangolo pieno con angoli arrotondati
PTSIN (0) = Prima ascissa
PTSIN (1) = Prima ordinata
PTSIN (2) = Seconda ascissa
PTSIN (3) = Seconda ordinata
.
.
.
PTSIN (n—1) = Ultima ascissa
PTSIN (n) = Ultima ordinata
INTIN (0) = Punto iniziale per archi o settori (misurati in decimi di grado)
INTIN (1) = Punto finale per archi o settori (misurati in decimi di grado)

GDP 1: Rettangolo

Il rettangolo pieno è probabilmente l'oggetto GDP più semplice; specificate le coordinate dell'angolo in alto a sinistra e di quello in basso a destra, quindi usate l'istruzione VDISYS per farlo comparire. Se volete che il rettangolo sia di qualche particolare colore, usate l'istruzione COLOR per cambiare il colore della linea.

Celle di memoria

CONTRL (0)=11
CONTRL (1)=2
CONTRL (3)=0
CONTRL (5)=1

PTSIN (0)=Ascissa dell'angolo in alto a sinistra
PTSIN (1)=Ordinata dell'angolo in alto a sinistra
PTSIN (2)=Ascissa dell'angolo in basso a destra
PTSIN (3)=Ordinata dell'angolo in basso a destra

Esempio di programma Il prossimo programma disegna un grafico ad istogrammi a colori impiegando la routine del rettangolo. Prima il computer cancella lo schermo, reimposta il generatore di numeri casuali e disegna le linee orizzontali del grafico:

```
 5 REM Grafico ad istogrammi a colori
 7 DEF SEG = 0
10 FULLW 2: CLEARW 2: RANDOMIZE 5
20 FOR Y=0 TO 160 STEP 10
30 LINEF 0,Y,300,Y
40 NEXT: COLOR 1,1,1
```

Poi il programma imposta la primitiva grafica ed inizia il loop che disegna l'istogramma sullo schermo (iniziando all'ascissa 10, spostandosi con incrementi di 20, finendo a 270).

```
50 POKE CONTRL,11
60 POKE CONTRL+2,2
70 POKE CONTRL+6,0
80 POKE CONTRL+10,1
90 FOR X=10 TO 270 STEP 20
```

Ora l'istruzione POKE inserisce il valore di X come ascissa dell'angolo in alto a destra, un numero casuale (da 20 a 140) per l'ordinata dell'angolo in alto a sinistra, X+10 per l'ascissa dell'angolo in basso a destra e 160 (il fondo del grafico) per la rispettiva ordinata.

Questo disegna un istogramma largo 10 pixel, che inizia sul fondo del grafico e prosegue fino ad un punto casuale.

```
100 POKE PTSIN,X
110 POKE PTSIN+2,RND*120+20
120 POKE PTSIN+4,X+10
130 POKE PTSIN+6,160
140 COLOR 1,RND*15,1
150 VDISYS (1)
160 NEXT
170 GOTO 170
```

GDP 2 e 3: Arco e settore

Se volete disegnare solo una parte di circonferenza, GDP 2 traccia l'arco iniziando ed interrompendo agli angoli che specificate. Tenete presente che gli angoli sono in decimi di grado, quindi se volete che l'arco inizi a 100 gradi e finisca a 164.5 gradi, impiegherete le cifre 1000 e 1645 (per sapere il valore di un grado rispetto all'intero cerchio, vedi l'istruzione CIRCLE nel Capitolo 4).

GDP 3 funziona nello stesso identico modo, salvo che traccia un settore di cerchio pieno, come una fetta di torta. GDP 3 trova probabilmente l'applicazione più utile in un programma che traccia diagrammi circolari.

Celle di memoria

CONTRL (0)=11
CONTRL (1)=4
CONTRL (3)=2
CONTRL (5)=2 per l'arco, 3 per il settore
INTIN (0)=Punto iniziale (in decimi di grado)
INTIN (1)=Punto finale (in decimi di grado)
PTSIN (0)=Ascissa del centro
PTSIN (1)=Ordinata del centro
PTSIN (2), (3), (4), (5) e (7)=0
PTSIN (6)=Raggio (in pixel)

Esempio di programma Il programma seguente disegna settori circolari pieni casuali aventi tutti lo stesso centro e lo stesso raggio; il risultato è un effetto vertiginoso di settori colorati che a tratti sembrano ruotare nel cerchio che i settori casuali formano. Innanzitutto gli elementi della routine vengono messi in memoria con istruzioni POKE:

```
 4 REM Settori pieni casuali
 7 DEF SEG = 0
10 FULLW 2: CLEARW 2
20 POKE CONTRL,11
30 POKE CONTRL+2,4
40 POKE CONTRL+6,2
50 POKE CONTRL+10,3
60 POKE PTSIN,150
70 POKE PTSIN+2,90
80 FOR X=2 TO 7: POKE PTSIN+(X*2),0: NEXT
90 POKE PTSIN+12,50
```

La parte del programma che disegna i settori casuali non fa altro che mettere due valori in memoria: l'angolo iniziale e quello finale del settore. VDISYS (1) disegna il settore, l'istruzione COLOR cambia il colore del settore in modo casuale e GOTO 100 reinizia la routine. Notate che le linee 100 e 110 contengono entrambe la formula *100*(RND*36)* per ottenere un angolo casuale compreso tra 100 e 3600:

```
100 POKE INTIN,100*(RND*36)
110 POKE INTIN+2,100*(RND*36)
120 VDISYS (1): COLOR 1,RND*15+1
130 GOTO 100
```

GDP 4: Cerchio pieno

Celle di memoria

CONTRL (0)=11
CONTRL (1)=3
CONTRL (3)=0
CONTRL (5)=4
PTSIN (0)=Ascissa del centro
PTSIN (1)=Ordinata del centro
PTSIN (2), (3) e (5)=0
PTSIN (4)=Raggio (misurato in pixel)

Esempi di programma Il prossimo programma stampa cerchi pieni verdi in sequenza regolare usando due loop FOR...NEXT annidati ed inserendo i valori da questi loop nelle aree PTSIN.

```
  4 REM Cerchi pieni verdi
  7 DEF SEG = 0
 10 FULLW 2: CLEARW 2
 20 COLOR 1,3,1
 30 POKE CONTRL,11
 40 POKE CONTRL+2,3
 50 POKE CONTRL+6,0
 60 POKE CONTRL+10,4
 70 FOR X=25 TO 575 STEP 50
 80 FOR Y=50 TO 150 STEP 25
 90 POKE PTSIN,X
100 POKE PTSIN+2,Y
```

```
110 POKE PTSIN+8,10
120 VDISYS (1)
130 NEXT Y: NEXT X
```

Vediamo un esempio che stampa cerchi a caso e cambia il colore dopo ciascuna istruzione VDISYS:

```
  4 REM Cerchi casuali
  7 DEF SEG = 0
 10 FULLW 2: CLEARW 2
 20 POKE CONTRL,11
 30 POKE CONTRL+2,3
 40 POKE CONTRL+6,0
 50 POKE CONTRL+10,4
 60 POKE PTSIN,150
 70 POKE PTSIN+2,90
 80 FOR X=2 TO 5: POKE PTSIN+(X*2),0: NEXT
 90 POKE PTSIN+8,RND*70
100 VDISYS (1): COLOR 1,RND*15+1
110 GOTO 90
```

Usando lo stesso centro per diversi cerchi colorati, potete creare un'immagine che assomiglia ad un bersaglio. Questo programma stampa otto cerchi, ciascuno di colore diverso, ma con lo stesso centro. Notate che il loop va a ritroso da 8 a 1; questo serve a conservare i cerchi man mano li disegnate, poiché se li tracciaste a partire dal più piccolo, ciascun cerchio verrebbe coperto dal successivo.

```
  5 REM Bersaglio
 10 FULLW 2: CLEARW 2
 20 POKE CONTRL,11
 30 POKE CONTRL+2,3
 40 POKE CONTRL+6,0
 50 POKE CONTRL+10,4
 60 POKE PTSIN,150
 70 POKE PTSIN+2,75
 80 FOR C=8 TO 1 STEP —1
 90 COLOR 1,C
100 POKE PTSIN+8,C*7
110 VDISYS (1)
120 NEXT
130 GOTO 130
```

GDP 5: Ellisse piena

L'ellisse richiede due raggi, uno sull'asse delle ascisse e l'altro sull'asse delle ordinate, e poiché GDP 5 disegna un'ellisse piena, usate l'istruzione COLOR per stabilire di quale colore volete che sia.

Celle di memoria

CONTRL (0)=11
CONTRL (1)=2
CONTRL (3)=0
CONTRL (5)=5
PTSIN (0)=Ascissa del centro
PTSIN (1)=Ordinata del centro
PTSIN (2)=Raggio sull'asse delle ascisse
PTSIN (3)=Raggio sull'asse delle ordinate

Esempi di programma Il primo programma relativo a GDP 5 traccia ellissi a caso sullo schermo usando un punto centrale costante, ma cambiando sia il raggio che il colore ad ogni nuova ellisse. Come potete vedere nella Figura

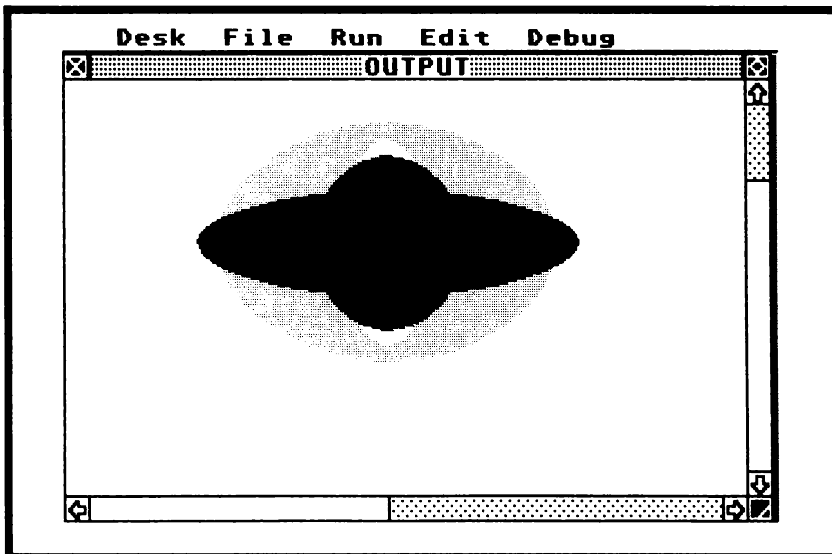


Figura 6.4: ellissi casuali, sovrapposte le une alle altre.

6.4, man mano che le ellissi si sovrappongono, si va formando un disegno interessante.

```
  5 REM Ellissi casuali sovrapposte
 10 FULLW 2: CLEARW 2
 20 POKE CONTRL,11
 30 POKE CONTRL+2,2
 40 POKE CONTRL+6,0
 50 POKE CONTRL+10,5
 60 POKE PTSIN,145
 70 POKE PTSIN+2,87
 80 POKE PTSIN+4,RND*100
 90 POKE PTSIN+6,RND*50
100 VDISYS (1)
105 COLOR 1,RND*15,RND*15
110 GOTO 80
```

Il prossimo programma è un po' più ordinato; anche se il centro rimane costante, i raggi cambiano gradualmente in modo che la prima ellisse piena è alta e stretta, mentre l'ultima è larga e bassa. Le altre 48 ellissi sono figure intermedie che si formano con il graduale aumento del raggio sull'asse delle

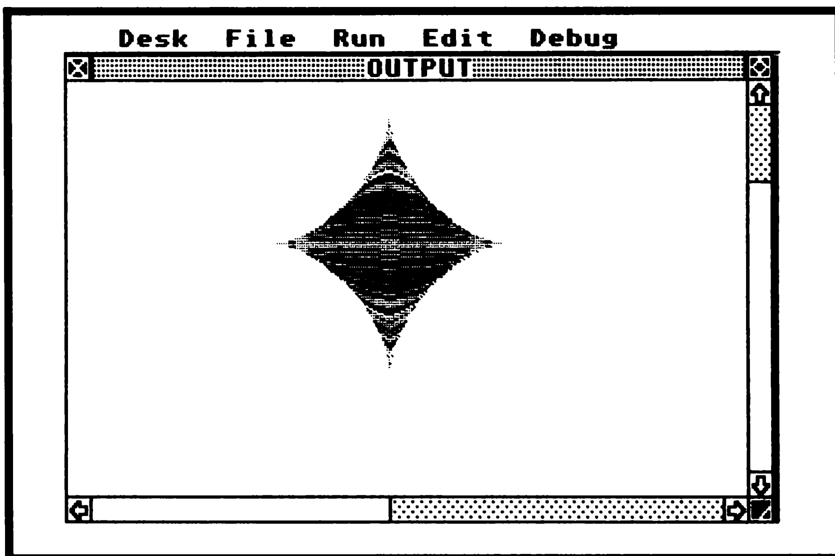


Figura 6.5: l'occhio di Dio.

X e della progressiva diminuzione del raggio sull'asse delle Y. Poiché i colori sono casuali per ogni ellisse, l'immagine che compare nella Figura 6.5 viene talvolta chiamata occhio di Dio.

```
  5 REM L'occhio di Dio
 10 FULLW 2: CLEARW 2
 20 POKE CONTRL,11
 30 POKE CONTRL+2,2
 40 POKE CONTRL+6,0
 50 POKE CONTRL+10,5
 60 POKE PTSIN,145
 70 POKE PTSIN+2,87
 80 FOR R=1 TO 50
 90 POKE PTSIN+4,R: POKE PTSIN+6,51—R
100 VDISYS (1)
105 COLOR 1,RND*16
110 NEXT
```

La linea 90 costituisce il cuore del programma, dato che il computer usa la variabile R del loop FOR...NEXT per determinare entrambi i raggi. Il raggio sull'asse delle X è uguale a R, il che significa che aumenterà da 1 a 50; quello sull'asse delle Y è 51—R, che significa che si accorcerà da 50 a 1.

GDP 6 e 7: Arco e settore di ellisse

Archi e settori ellittici sono simili agli archi e ai settori circolari, compreso il fatto che gli angoli iniziale e finale vengono misurati in decimi di grado. Le uniche differenze, naturalmente, riguardano le misure dei due raggi, la cui differenza vi permette di disegnare archi e settori irregolari.

Celle di memoria

CONTRL (0)=11
CONTRL (1)=2
CONTRL (2)=2
CONTRL (5)=6 per arco di ellisse, 7 per settore di ellisse
INTIN (0)=Angolo iniziale dell'arco o del settore
INTIN (1)=Angolo finale dell'arco o del settore
PTSIN (0)=Ascissa del centro
PTSIN (1)=Ordinata del centro

PTSIN (2) = Raggio sull'asse Xdelle ascisse
PTSIN (3) = Raggio sull'asse Ydelle ordinate

Esempi di programma Vediamo un esempio molto semplice di come funziona questa routine. Innanzitutto, si danno le normali istruzioni POKE:

```
 5 REM Archi ellittici
10 FULLW 2: CLEARW 2
20 POKE CONTRL,11
30 POKE CONTRL+2,2
40 POKE CONTRL+4,2
50 POKE CONTRL+10,6
```

Per ottenere un raggio casuale per l'ellisse, il computer seleziona un'ascissa casuale tra 30 e 250 (linea 60) e un'ordinata casuale tra 30 e 150 (linea 70). Alla linea 80 viene selezionato un raggio in ascissa casuale compreso tra 1 e 30 e nella linea 90 un raggio in ordinata compreso tra 1 e 25. Infine gli angoli iniziale e finale vengono creati nelle linee 100-110 e l'arco viene visualizzato con l'istruzione VDISYS (1) della linea 120.

```
 60 POKE PTSIN,RND*220+30
 70 POKE PTSIN+2,RND*120+30
 80 POKE PTSIN+4,RND*30
 90 POKE PTSIN+6,RND*25
100 POKE INTIN,(RND*36)*100
110 POKE INTIN+2,(RND*36)*100
120 VDISYS (1): COLOR 1,1,RND*16
130 GOTO 60
```

Il prossimo programma assomiglia al precedente, ma traccia settori ellittici (che sono pieni, anziché essere semplici archi). Il risultato compare nella Figura 6.6.

```
 5 REM Settori ellittici casuali
10 FULLW 2: CLEARW 2
20 POKE CONTRL,11
30 POKE CONTRL+2,2
40 POKE CONTRL+4,2
50 POKE CONTRL+10,7
60 POKE PTSIN,RND*220+30
70 POKE PTSIN+2,RND*120+30
80 POKE PTSIN+4,RND*30
```

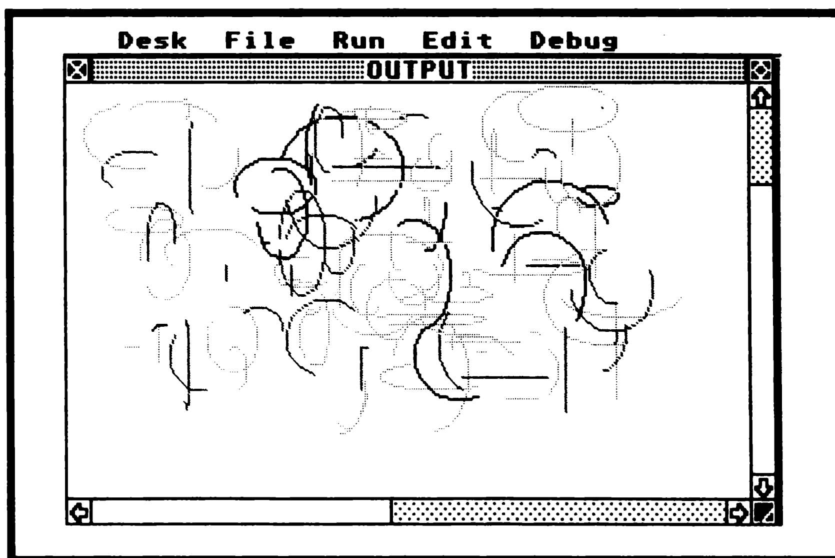


Figura 6.6: archi ellittici casuali.

```
 90 POKE PTSIN+6,RND*25
100 POKE INTIN,(RND*36)*100
110 POKE INTIN+2,(RND*36)*100
120 VDISYS (1): COLOR 1,RND*16
130 GOTO 60
```

GDP 8 e 9: Rettangolo arrotondato e rettangolo pieno arrotondato

Queste ultime due funzioni GDP disegnano un rettangolo con gli angoli arrotondati, l'uno vuoto e l'altro pieno.

Celle di memoria

CONTRL (0)=11
CONTRL (1)=2
CONTRL (3)=0
CONTRL (5)=8 per il rettangolo arrotondato, 9 per il rettangolo pieno arrotondato
PTSIN (0)=Ascissa dell'angolo in alto a sinistra
PTSIN (1)=Ordinata dell'angolo in alto a sinistra

PTSIN (2)=Ascissa dell'angolo in basso a destra
PTSIN (3)=Ordinata dell'angolo in basso a destra

Esempi di programma Usando la funzione per il rettangolo arrotondato (GDP 8) e disegnandone un certo numero nella stessa area, potete creare il disegno della Figura 6.7, che ricorda la veduta aerea di una piramide. L'effetto tridimensionale è interessante e con questa funzione VDISYS, è anche facile da produrre.

```
  5 REM Veduta aerea di una piramide
  7 DEF SEG = 0
 10 FULLW 2: CLEARW 2
 20 POKE CONTRL,11
 30 POKE CONTRL+2,2
 40 POKE CONTRL+6,0
 50 POKE CONTRL+10,8
 60 FOR X=60 TO 130 STEP 4
 70 POKE PTSIN,X
 80 POKE PTSIN+2,X/2
 90 POKE PTSIN+4,250—X
100 POKE PTSIN+6,150—X/2
110 VDISYS (1)
120 NEXT
```

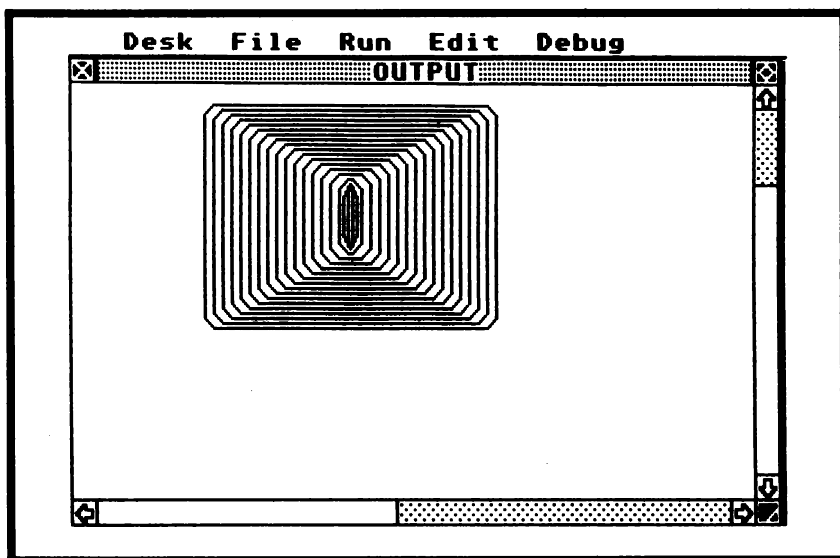


Figura 6.7: veduta aerea di una piramide.

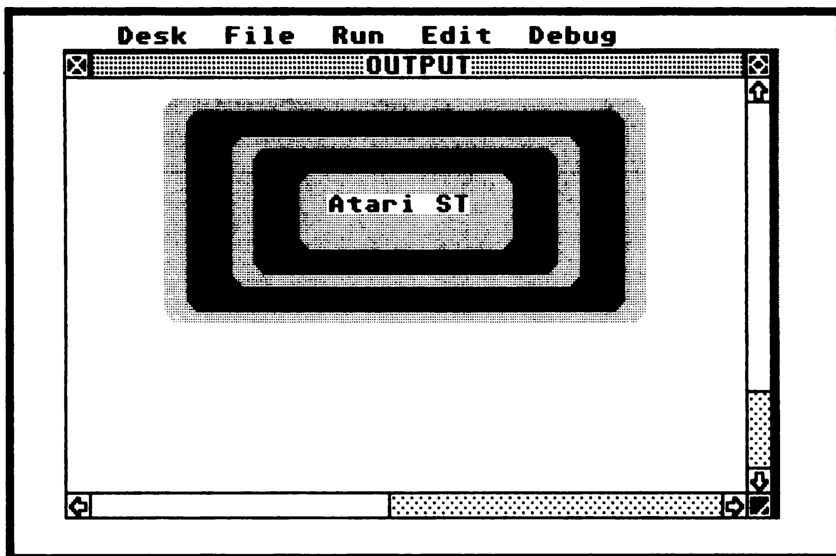


Figura 6.8: un'insegna ottenuta usando rettangoli pieni arrotondati.

Disegnando rettangoli pieni arrotondati che siano proporzionali tra loro, potete creare un'insegna a cornici concentriche con le parole *Atari ST* al centro, come nella Figura 6.8. Il programma inizia con delle istruzioni POKE per identificare la funzione che verrà impiegata.

```
 5 REM Insegna
10 FULLW 2: CLEARW 2
20 POKE CONTRL,11
30 POKE CONTRL+2,2
40 POKE CONTRL+6,0
50 POKE CONTRL+10,9
```

Ora il computer disegna sette rettangoli pieni (con gli angoli arrotondati) mettendo il valore variabile di X (usato in una variabile FOR...NEXT) nell'area 14 determinata da PTSIN. Le coordinate dell'angolo in alto a sinistra del rettangolo sono (X-15,X/2) e quelle dell'angolo in basso a destra sono (320-X,150—X/2). Man mano X va da 60 a 120 (con incrementi di 10), il rettangolo diventa sempre più piccolo e i rettangoli più grandi che sono già tracciati rimangono sullo schermo.

```
60 FOR X=60 TO 120 STEP 10
70 POKE PTSIN,X—15
```

```
 80 POKE PTSIN + 2,X/2
 90 POKE PTSIN + 4,320—X
100 POKE PTSIN + 6,150—X/2
110 VDISYS (1): COLOR 1,RND*16
120 NEXT: GOTOXY 13,5: PRINT "Atari ST"
130 GOTO 130
```

Ora che l'insegna colorata è stata completata, il computer va alla colonna 13, riga 5 e scrive *Atari ST.*

Codice di controllo 14: cambia il livello di rosso, verde e blu di un colore

Se trovate troppo complicata la procedura per cambiare il livello di rosso, verde e blu di un colore BASIC ST, probabilmente sarete felici di sapere che il Codice di Controllo 14 è molto più facile da usare di VARPTR. Se volete cambiare un colore memorizzato con un numero di colore, usate POKE INTIN con il numero del colore che volete cambiare e POKE INTIN + 2, POKE INTIN + 4 e POKE INTIN + 6 con i livelli di rosso, verde e blu che volete assegnare al nuovo colore (compresi tra 0 e 1000 per ciascun'area d'input). Dopo aver battuto *VDISYS (1),* al numero in uso corrisponderà il nuovo colore che avete specificato e lo potrete usare battendo *COLOR* seguito dallo stesso numero di colore.

Celle di memoria

CONTRL (0) = 14
CONTRL (1) = 0
CONTRL (3) = 4
INTIN (0) = Numero del colore che volete cambiare (vedi Tabella 4.1)
INTIN (1) = Intensità del livello di rosso (0-1000)
INTIN (2) = Intensità del livello di verde (0-1000)
INTIN (3) = Intensità del livello di blu (0-1000)

Esempi di programma Nel prossimo programma si disegna un cerchio pieno al centro dello schermo ed il suo colore (numero 2) viene cambiato con il Codice di controllo 14.

Notate che se anche il colore è già sullo schermo, vien cambiato istantaneamente con l'istruzione VDISYS.

```
  5 REM Cerchio pieno che cambia colore
  7 DEF SEG = 0
 10 FULLW 2: CLEARW 2
 20 COLOR 1,2,2
 30 PCIRCLE 150,75,50
 40 POKE CONTRL,14
 50 POKE CONTRL+2,0
 60 POKE CONTRL+6,4
 70 POKE INTIN,2
 80 POKE INTIN+2,RND*1000
 90 POKE INTIN+4,RND*1000
100 POKE INTIN+6,RND*1000
110 VDISYS (1): GOTO 80
```

Se avete abbastanza tempo per vedere i 512 colori che l'ST è capace di produrre, lanciate il programma che segue. Usando tre loop FOR...NEXT (con valori di STEP piuttosto alti in modo che non dobbiate aspettare troppo a lungo), il computer scorre tutti i livelli di colore.

Mentre il programma gira, i livelli di rosso, verde e blu vengono visualizzati nella parte alta dello schermo, in modo che possiate farvi un'idea del significato dei tre numeri:

```
  5 REM Cicli dei colore
  7 DEF SEG = 0
 10 FULLW 2: CLEARW 2
 20 COLOR 1,2,2
 30 PCIRCLE 150,75,50
 40 POKE CONTRL,14
 50 POKE CONTRL+2,0
 60 POKE CONTRL+6,4
 70 POKE INTIN,2
 80 FOR R=0 TO 1000 STEP 50
 90 FOR G=0 TO 1000 STEP 50
100 FOR B=0 TO 1000 STEP 50
110 POKE INTIN+2,R
120 POKE INTIN+4,G
130 POKE INTIN+6,B
140 GOTOXY 0,0: PRINT R,G,B
150 VDISYS (1): NEXT: NEXT: NEXT: CLEARW 2
```

Codice di controllo 15: cambia il tipo di linea disegnato

L'istruzione BASIC LINEF e la funzione polyline VDISYS tracciano sullo schermo linee sottili e continue, ma con questa funzione potete modificare il tipo di linea tracciata. Si usa l'istruzione POKE per mettere nella cella di memoria INTIN il numero corrispondente al tipo di linea che volete e più avanti in questo capitolo imparerete come definire dei vostri tipi di linea (a cui potete accedere usando *POKE INTIN,7* nel programma).

Celle di memoria

CONTRL (0) = 15
CONTRL (1) = 0
CONTRL (3) = 1
INTIN (0) = Tipi di linea:
1 Linea continua
2 Lineette lunghe
3 Linea punteggiata
4 Lineetta punto
5 Lineette

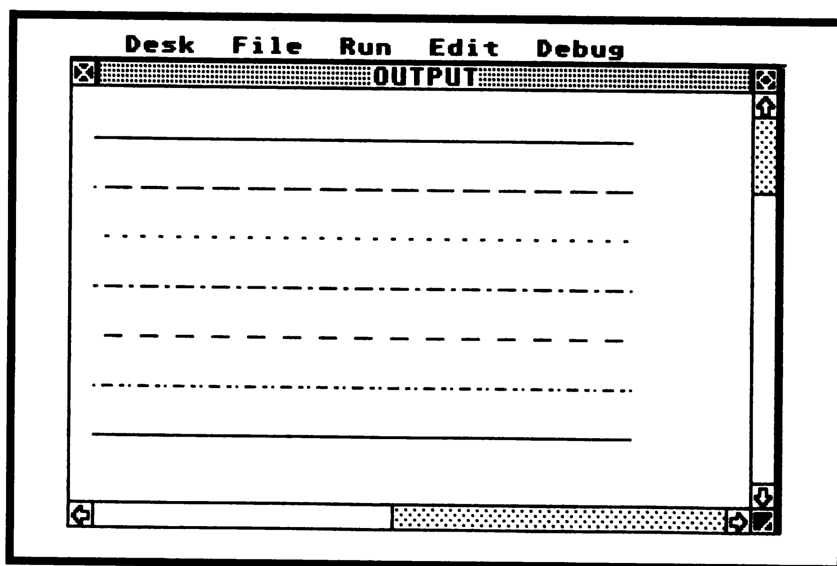


Figura 6.9: tipi di linea programmabili.

6 Lineetta punto punto
7 Definita dall'utente (vedi Codice di Controllo 113)

Esempi di programma I sei diversi tipi di linea memorizzati sono presentati nella Figura 6.9, che si riproduce lanciando il seguente programma:

```
 5 REM Tipi di linea programmabili
 7 DEF SEG = 0
10 FULLW 2: CLEARW 2
20 POKE CONTRL,15
30 POKE CONTRL+2,0
40 POKE CONTRL+6,1
45 FOR C=1 TO 6
50 POKE INTIN,C
60 VDISYS (1)
70 LINEF 10,10,250,100
80 FOR D=1 TO 500: NEXT: NEXT
```

Codice di controllo 16: imposta la larghezza delle linee tracciate

L'ST vi offre ulteriore flessibilità nel disegnare le linee permettendovi di cambiare la larghezza delle linee che userete. L'istruzione POKE opera sull'area di memoria PTSIN con la larghezza che volete. Usate VDISYS in modo che ogni altra linea che disegnerete sarà della nuova larghezza.

Celle di memoria

CONTRL (0)=16
CONTRL (1)=1
CONTRL (3)=0
PTSIN (0)=Larghezza della linea disegnata (misurata in pixel; default=1)
PTSIN (1)=0

Esempi di programma Quello che segue è un breve programma che mostra linee orizzontali di diversa larghezza (e di diversi colori) disegnati dall'alto in basso dello schermo. Il numero del colore di ciascuna è uguale anche alla larghezza (la linea larga tre pixel è del colore numero 3, per esempio) e quello

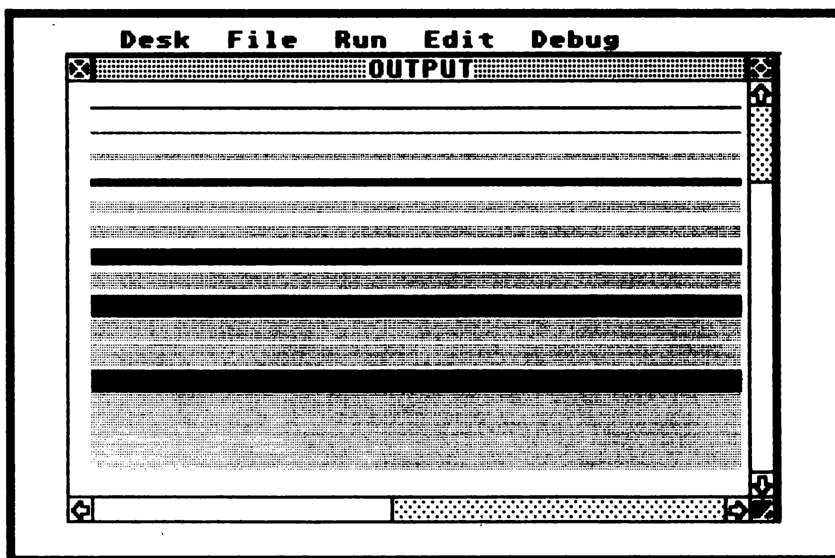


Figura 6.10: come cambiare la larghezza delle linee.

che risulta sullo schermo, come nella Figura 6.10, assomiglia un po' ad un test di colore su un televisore.

```
  5 REM Larghezza delle linee variabile
  7 DEF SEG = 0
 10 FULLW 2: CLEARW 2
 20 POKE CONTRL,16
 30 POKE CONTRL+2,1
 40 POKE CONTRL+6,0
 50 FOR W=1 TO 15: COLOR 1,1,W
 60 POKE PTSIN,W
 70 VDISYS (1)
 80 LINEF 10,W*10,300,W*10
 90 NEXT
100 GOTO 100
```

Codice di controllo 17: imposta il colore delle linee

Vediamo una terza funzione di gestione delle linee, che vi permette di sceglierne il colore. Potreste usare l'istruzione COLOR, ma se volete una funzio-

ne più "diretta" che faccia parte della chiamata VDI, questa produce lo stesso effetto.

Celle di memoria

CONTRL (0)=17
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Colore che volete assegnare alla linea (vedi Tabella 4.1)

Codice di controllo 18: imposta il tipo di polymarker

Avete già imparato come collocare i polymarker in punti specifici dello schermo con il Codice di Controllo 6. Il polymarker tuttavia non deve essere necessariamente un pixel singolo e questa funzione vi permette di cambiare il polymarker; rimarrà comunque centrato nel punto da voi specificato.

Celle di memoria

CONTRL (0)=18
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Tipo di polymarker
1 Punto
2 Croce
3 Asterisco
4 Quadratino
5 Croce diagonale
6 Rombo

Esempi di programma Se volete vedere i sei diversi tipi di polymarker visualizzati a caso, date un'occhiata a questo programma. Le prime linee iniziano il loop FOR...NEXT che scorre i sei diversi tipi ed impiega l'istruzione POKE per mettere in memoria i valori che servono a cambiare il tipo in uso:

```
 5 REM I sei tipi di polymarker
 7 DEF SEG = 0
10 FULLW 2: CLEARW 2: FOR T=1 TO 6
20 POKE CONTRL,18
30 POKE CONTRL+2,0
```

```
40 POKE CONTRL+6,1
50 POKE INTIN,T: VDISYS (1)
```

Poi si ricorre al Codice di Controllo 7 per visualizzare il tipo di polymarker in uso in nove posizioni casuali dello schermo. L'ascissa può variare da 1 a 300, mentre l'ordinata può variare da 30 a 150; il valore minimo dell'ordinata è 30 per impedire che il polymarker venga visualizzato troppo in alto, al di sopra della finestra in uso.

```
  0 POKE CONTRL,7
 70 POKE CONTRL+2,5
 80 POKE CONTRL+6,0
 90 FOR I=1 TO 9 STEP 2
100 POKE PTSIN+(I-1)*2,RND*300
110 POKE PTSIN+I*2,RND*120+30
120 NEXT
130 VDISYS (1)
140 NEXT
```

Dato che il computer non cancella la finestra Output tutte le volte che visualizza un polymarker nuovo, risulta che, come nella Figura 6.11, compaiono sullo schermo i sei tipi di polymarker.

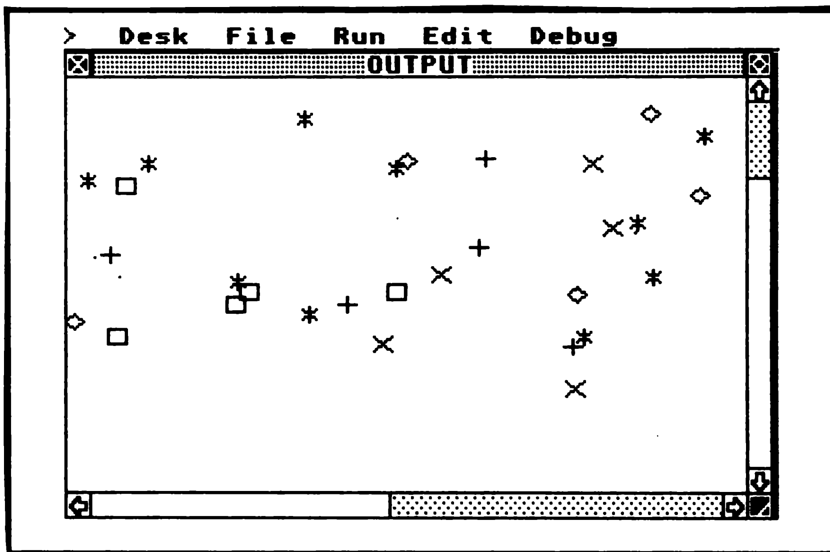


Figura 6.11: i sei tipi di polymarker.

Codice di controllo 20: imposta il colore dei polymarker

L'ultima funzione relativa ai polymarker è il Codice di Controllo 20, che cambia il colore di polymarker successivi.

Celle di memoria

CONTRL (0)=20
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Colore che volete assegnare al polymarker (vedi Tabella 4.1)

Codici di controllo 23 e 24: impostano un motivo di colorazione del fondo

I Codici di controllo 23 e 24 funzionano insieme per dire al computer quale motivo deve impiegare per la routine di colorazione del fondo. Se non siete sicuri di come si determinino i motivi di colorazione del fondo, consultate la descrizione dell'istruzione FILL del BASIC ST nel Capitolo 4.

Una volta che conoscete i due numeri per la colorazione del fondo che volete, inserite il primo con il Codice di Controllo 23 ed il secondo con il Codice di Controllo 24; poiché entrambi i codici (e le rispettive istruzioni POKE) sono indispensabili per determinare il fondo, dovrete impiegare due chiamate VDISYS, una dopo aver completato l'input per il Codice 23 e una dopo il Codice 24.

Celle di memoria

Prima parte

CONTRL (0)=23
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Stabilisce il motivo di colorazione del fondo che intendete usare:
0 Vuoto (nessuna coloritura)
1 Pieno
2 Motivo
3 Tratteggio

CONTRL (0)=24
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Da 1 a 24 per i motivi e da 1 a 12 per i tratteggi, come con il comando BASIC COLOR (vedi Figura 4.6).

Esempio di programma Vediamo un breve programma che disegna un cerchio pieno impiegando il motivo di colorazione del fondo 2,24.

```
 5 FULLW 2: CLEARW 2
10 POKE CONTRL,23
20 POKE CONTRL+2,0
30 POKE CONTRL+6,1
40 POKE INTIN,2: VDISYS (1)
50 POKE CONTRL,24
60 POKE CONTRL+2,0
70 POKE CONTRL+6,1
80 POKE INTIN,9
90 PCIRCLE 150,75,30
```

Numero	**Colore**
0	bianco
1	nero
2	rosso
3	verde
4	azzurro
5	ottanio
6	giallo
7	magenta
8	grigio
9	nero
10	rosso chiaro
11	verde chiaro
12	azzurro chiaro
13	ottanio chiaro
14	giallo chiaro
15	magenta chiaro

Tabella 6.1: colori di fondo.

Codice di controllo 25: imposta il colore di una forma piena

Userete il Codice di Controllo 25 se volete cambiare il colore impiegato nella routine di riempimento (se state usando l'istruzione FILL o qualcosa simile a PCIRCLE); potrete altrettanto facilmente specificare il colore di fondo con il secondo argomento di COLOR.

Celle di memoria

CONTRL (0) = 25
CONTRL (1) = 0
CONTRL (3) = 1
INTIN (0) = Imposta il colore che volete per il fondo (vedi Tabella 6.1)

Codice di controllo 103: colorazione del fondo di un'area specificata

La funzione che riempie un'area specificata è identica all'istruzione FILL del BASIC ST salvo un'importante differenza: potete dire al computer a quale bordo di un determinato colore deve smettere di disegnare. Supponiamo, per esempio, di avere diversi cerchi concentrici di diversi colori e di dare un'istruzione FILL per iniziare a riempire di colore partendo dal centro; con il colore specificato con la più recente istruzione COLOR, verrà riempito solo il cerchio più interno. Il Codice di controllo 103, invece, comincia a colorare dal centro verso l'esterno, in modo da riempire tutta l'area interna al cerchio del colore da voi specificato. Questa funzione VDISYS è utile quando volete riempire solo una certa area e tale area è completamente delimitata da un certo colore (che il colore appartenga ad un quadrato, ad un cerchio o ad una qualsiasi forma).

Celle di memoria

CONTRL (0) = 103
CONTRL (1) = 1
CONTRL (3) = 1
INTIN (0) = Numero del colore dove volete che il computer smetta di colorare; vedi Tabella 6.1
PTSIN (0) = Ascissa del punto in cui volete cominciare a riempire di colore
PTSIN (1) = Ordinata del punto in cui volete cominciare a riempire di colore

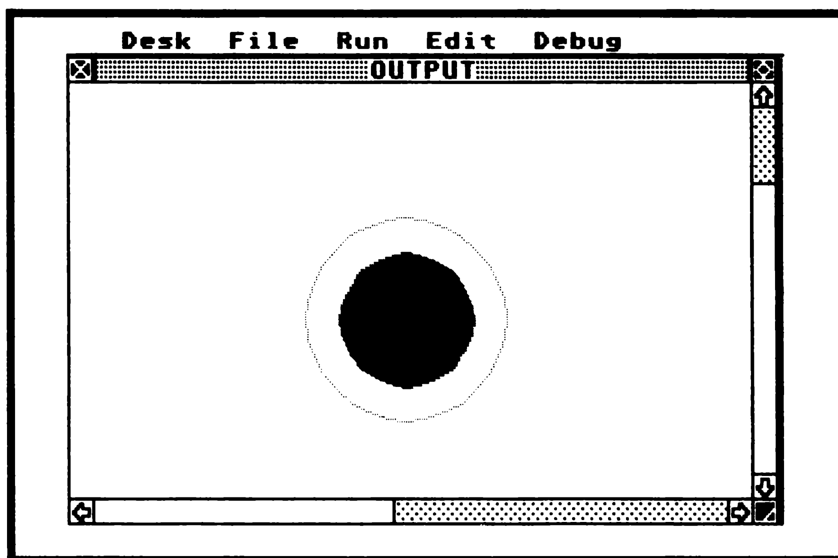


Figura 6.12: riempimento di colore solo dei due cerchi più interni.

Esempi di programma Il prossimo programma è simile a quello descritto nel paragrafo precedente: disegna tre cerchi concentrici (ciascuno di diverso colore) e i primi due vengono riempiti con il Codice di Controllo 103 e VDISYS. L'istruzione POKE INTIN,2 nella linea 90 dice al computer di smettere di disegnare quando raggiunge il colore numero 2 e siccome il secondo cerchio è proprio di questo colore, l'intera area entro il secondo cerchio (e di conseguenza anche quella all'interno del primo cerchio) viene riempita. La Figura 6.12 mostra il risultato.

```
  5 REM Colorazione del fondo solo dei due cerchi interni
 10 FULLW 2: CLEARW 2
 20 FOR C=3 TO 1 STEP —1
 30 COLOR 1,1,C
 40 CIRCLE 150,95,C*15
 50 NEXT
 60 POKE CONTRL,103
 70 POKE CONTRL+2,1
 80 POKE CONTRL+6,1
 90 POKE INTIN,2
100 POKE PTSIN,150
110 POKE PTSIN+2,95
120 VDISYS (1)
```

Codice di controllo 108: fissa la forma delle estremità di una linea

Avete già imparato come cambiare il tipo di polymarker in modo da avere più flessibilità nell'impiegarli. Il Codice di controllo 108 aumenta ulteriormente tale flessibilità, poiché vi permette di cambiare la forma delle estremità di una linea. L'impostazione di default è semplicemente un punto, che non dà all'estremità della linea alcun aspetto particolare. Impiegando questo codice, potete invece trasformare le estremità delle linee in frecce, oppure arrotondarle.

Celle di memoria

CONTRL (0)=108
CONTRL (1)=0
CONTRL (3)=2
INTIN (0)=Forma da usare per l'estremità iniziale della linea:
0 Estremità squadrata (impostazione di default)
1 Freccia
2 Arrotondata
INTIN (1)=Forma da usare per l'estremità finale della linea (si usano gli stessi numeri impiegati per l'estremità iniziale della linea)

Codice di controllo 112: colorazione di fondo definita dall'utente

Se volete programmare la vostra colorazione di fondo, potete crearne una con un massimo di 16 parole di due byte, con il bit 15 della parola 1 nell'angolo in alto a sinistra del motivo e il bit 0 della parola 16 nell'angolo in basso a destra, come si vede nella Figura 6.13. Tuttavia potete anche fare motivi più semplici; supponiamo che vogliate un fondo simile ad una scacchiera. Basterà usare due parole per dire al computer lo stile che volete. La prima parola avrà il primo bit acceso, il secondo spento, il terzo acceso e così via. La seconda parola sarà rovesciata rispetto alla prima, con il primo bit spento, il secondo acceso, il terzo spento e così via. In binario (dove un pixel acceso corrisponde a 1 e un pixel spento corrisponde a 0) le due parole si presenteranno così:

```
1010101010101010
0101010101010101
```

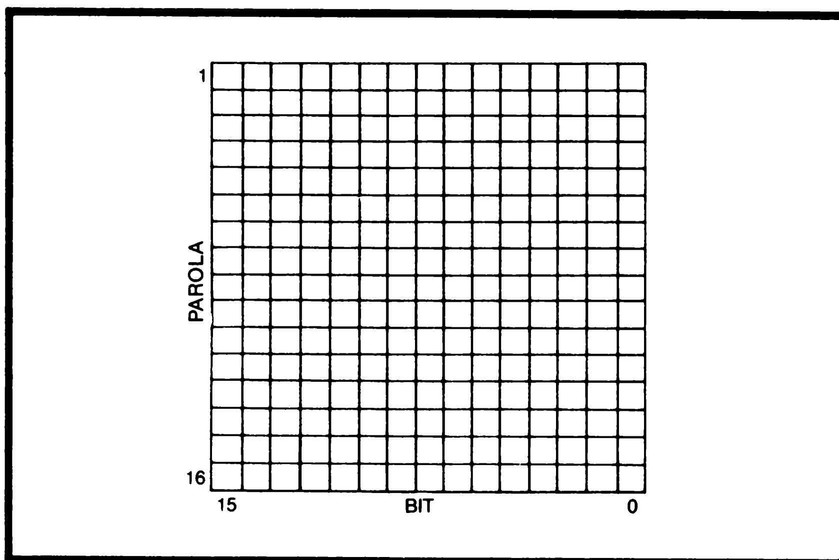


Figura 6.13: un reticolo per creare motivi di fondo.

Traducendo questi numeri binari in decimali, il primo è uguale a 43690 ed il secondo a 21845. Questi due numeri saranno la base per il vostro motivo di fondo ed oltre ad impiegare le celle di memoria elencate di seguito, userete POKE per inserire 43690 in INTIN e 21845 in INTIN+2. Per attivare questo motivo creato dall'utente, userete il Codice di Controllo 23 in modo che il computer sappia di usare lo stile 4 *(POKE INTIN,4).* Potete fare motivi più complicati impiegando fino a 16 set di numeri; prestate tuttavia attenzione quando calcolate i numeri decimali da quelli binari, poiché è molto facile commettere degli errori.

Celle di memoria

CONTRL (0)=112
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Primo dato del motivo
INTIN (1)=Secondo dato del motivo
.
.
.
INTIN (15)=Ultimo dato del motivo

Codice di controllo 113: imposta il motivo per il tipo di linea

Potete anche stabilire un vostro motivo per le linee che disegnate. Avete imparato prima a fare le linee punteggiate, linee costituite da punti e trattini ed alcune altre con il Codice di controllo 15. Se vi servono altri tipi di linea, dovete individuare quali pixel della linea sono *accesi* (che significa che il bit è uguale a 1) e quali non lo sono (che significa che il bit è uguale a 0). Supponiamo che vogliate formare una linea costituita da punti alternati a pixel vuoti; i 16 bit del suo codice binario saranno come segue:

```
0101010101010101
```

Questo numero binario è uguale al decimale 21845, quindi dovrete specificare *POKE INTIN,21845* oltre ad usare le celle di memoria elencate qui di seguito. Se volete tornare alla linea continua, usate il motivo

```
1111111111111111
```

che è uguale al decimale 65535.

Celle di memoria

CONTRL (0)=113
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Tipo di linea (vedi Codice di controllo 15)

Codice di controllo 114: rettangolo pieno

Il BASIC ST non ha un'istruzione interna per il rettangolo, ma il Codice di controllo 114 può servire allo scopo, poiché può disegnare un rettangolo in qualsiasi punto dello schermo. Il suo colore è determinato dal colore di fondo stabilito in memoria, quindi, usando le istruzioni POKE per stabilirne l'angolo in alto a sinistra e quello in basso a destra, potete far comparire il rettangolo con l'istruzione VDISYS (1).

Celle di memoria

CONTRL (0)=114
CONTRL (1)=2
CONTRL (3)=0
PTSIN (0)=Ascissa dell'angolo in alto a sinistra del rettangolo
PTSIN (1)=Ordinata dell'angolo in alto a sinistra del rettangolo
PTSIN (2)=Ascissa dell'angolo in basso a destra del rettangolo
PTSIN (3)=Ordinata dell'angolo in basso a destra del rettangolo

Esempio di programma Il prossimo programma è un esempio eccellente del potenziale grafico di questa semplice routine. Il programma disegna semplicemente diversi rettangoli colorati, ma posiziona ciascuno di essi leggermente più in basso a destra rispetto al precedente; ciò produce un fantastico effetto tridimensionale. Innanzitutto vengono messe in memoria le istruzioni POKE per questo codice di controllo:

```
 5 REM Un disegno tridimensionale
 7 DEF SEG = 0
10 FULLW 2: CLEARW 2
20 POKE CONTRL,114
30 POKE CONTRL+2,2
40 POKE CONTRL+6,0
```

Il loop FOR...NEXT alla linea 50 inizierà a spostare i rettangoli in basso a destra poiché sia ascissa che ordinata sono aumentate del valore di X man mano che esso aumenta:

```
50 FOR X=20 TO 130
60 POKE PTSIN,X
70 POKE PTSIN+2,X
80 POKE PTSIN+4,X+40
90 POKE PTSIN+6,X+20
```

Infine VDISYS (1) disegna tutti i rettangoli sullo schermo ed il computer ne cambia a caso il colore e continua il loop; una volta che il loop è completato, il GOTO 130 nella linea 130 blocca l'immagine sullo schermo in modo che

possiate guardarla. Lanciate il programma per vedere l'effetto tridimensionale.

```
100 VDISYS (1)
110 COLOR 1,RND*15+1,RND*15+1
120 NEXT
130 GOTO 130
```

Codice di controllo 124: controllore del mouse

Il controllore del mouse impiega aree di memoria per le quali potete ricorrere a PEEK per avere informazioni utili. Il Codice di Controllo 124 vi può dire la situazione del puntatore del mouse sullo schermo e la posizione dei due bottoni del mouse. Per esempio, PTSOUT (1) contiene l'attuale ordinata del puntatore del mouse; per memorizzarla in una variabile chiamata Y, basta battere:

```
Y=PEEK (PTSOUT+2)
```

Potete sfruttare le informazioni sul mouse in molte applicazioni, per scopi come disegnare, far selezionare un'opzione all'utente, o ricevere l'input in un videogioco.

Celle di memoria

CONTRL (0)=124
CONTRL (1)=0
CONTRL (3)=0

Output

INTOUT (0)=Stato del bottone del mouse:
0 Nessun bottone premuto
1 Bottone sinistro premuto
2 Bottone destro premuto
3 Entrambi i bottoni premuti
PTSOUT (0)=Ascissa del cursore grafico
PTSOUT (1)=Ordinata del cursore grafico

Esempio di programma Il programma seguente controlla costantemente il mouse e visualizza ascissa e ordinata nonché lo stato dei bottoni. Prima i numeri vengono messi in memoria con POKE e VDISYS (1) attiva la routine di controllo del mouse:

```
 5 REM La posizione del mouse cambia al mutare dei valori
 7 DEF SEG = 0
10 FULLW 2: CLEARW 2
20 POKE CONTRL,124
30 POKE CONTRL+2,0: POKE CONTRL+6,0
40 X=PEEK(PTSOUT): Y=PEEK(PTSOUT+2)
50 BS=PEEK(INTOUT)
60 VDISYS (1)
```

Ora il computer visualizza i valori in memoria; notate che il punto e virgola della linea 70 impedisce al computer di saltare alla linea successiva. L'istruzione della linea 80 fa andare il programma all'espressione appropriata a descrivere lo stato dei bottoni; BS+1 viene impiegato al posto di un semplice BS perché BS è uguale a 0 quando nessuno dei bottoni è premuto e ON GOSUB risponde solo per numeri uguali o maggiori di 1.

```
70 PRINT "X="x", Y="y", Bottone=";
80 ON BS+1 GOSUB 100,110,120,130
```

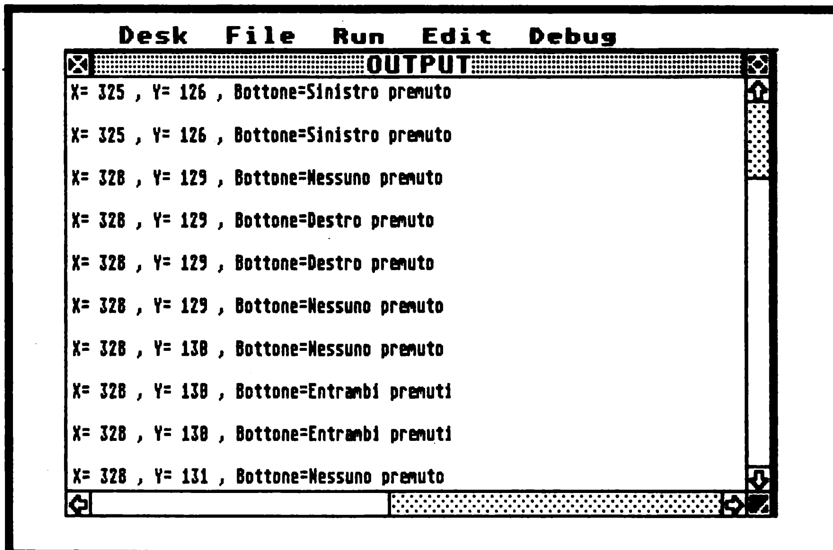


Figura 6.14: la posizione del mouse cambia al cambiare dei valori.

Quando il computer ritorna dalla appropriata routine, emette un punto di domanda nella linea 90 (che è l'equivalente di PRINT) per far saltare il programma alla linea successiva, poi ritorna alla linea 40 per ripetere la routine di controllo del mouse.

Le espressioni nelle linee da 100 a 130 vi dicono lo stato dei bottoni del mouse e emettono un RETURN nel punto del programma dove si trovava GOSUB. L'output compare nella Figura 6.14.

```
 90 ?: GOTO 40
100 PRINT "Nessuno premuto": RETURN
110 PRINT "Sinistro premuto": RETURN
120 PRINT "Destro premuto": RETURN
130 PRINT "Entrambi premuti": RETURN
```

CODICI DI CONTROLLO PER LE FUNZIONI DI ELABORAZIONE TESTI VDISYS

Finora avete imparato le funzioni grafiche del VDISYS. Ci sono anche delle operazioni sui testi nel GEM che vi permettono di usare una speciale modalità di testo, di visualizzare parole e numeri, di spostare il cursore per tutto lo schermo e di manipolare in altri modi un testo.

Codice di controllo 5: entra nella modalità testo

Si usa il Codice di Controllo 5 per entrare nella modalità di testo; ricordate che questa modalità non è come la normale istruzione PRINT del BASIC ST. È testo in grafica che può essere alterato e non funziona associato ad istruzioni PRINT.

Celle di memoria

CONTRL (0)=5
CONTRL (1)=0
CONTRL (3)=0
CONTRL (5)=3

Esempio di programma Questo breve programma entra nella modalità di testo, cancella la barra dei menù e porta il cursore nella parte alta dello schermo:

```
5 REM
10 FULLW 2: CLEARW 2
20 POKE CONTRL,5
30 POKE CONTRL+2,0
40 POKE CONTRL+6,0
50 POKE CONTRL+10,3
60 VDISYS (1)
```

I comandi di testo della Tabella 6.2 impiegano tutti le stesse istruzioni POKE come quelli elencati nella modalità Enter Text tranne che per CONTRL (5); sono elencati qui per vostra consultazione, ma ricordate di usare tutte le istruzioni POKE in aggiunta a POKE CONTRL + 10.

CONTRL (5) =	Comando
2	Esce dalla modalità di testo
4	Sposta il cursore verso l'alto
5	Sposta il cursore verso il basso
6	Sposta il cursore verso destra
7	Sposta il cursore verso sinistra
8	Porta il cursore nella sua posizione naturale, nell'angolo in alto a sinistra
9	Cancella il testo dalla posizione del cursore fino alla fine dello schermo
10	Cancella il testo dalla posizione del cursore fino alla fine della linea
11	Pone il cursore a riga e colonna specifiche (simile a GOTOXY): CONTRL (5) = 11 INTIN (0) = ascissa (numero della riga del cursore) INTIN (1) = ordinata (numero della colonna)
12	(Non usato)
13	Attiva il video in reverse
14	Disattiva il video in reverse
15	Dà l'attuale posizione del cursore: CONTRL (5) = 15 INTOUT (0) = Area di output della memoria dove è collocato il numero della riga INTOUT (1) = Area di output della memoria dove è collocato il numero della colonna

Tabella 6.2: comandi di testo usati con CONTRL (5).

Codice di controllo 21: seleziona la fonte del testo

Ci sono 17 *fonti* (stili tipografici del testo) a vostra disposizione nella modalità di testo grafico e potete inserire direttamente il numero della fonte che volete con il Codice di controllo 21.

Celle di memoria

CONTRL (0)=21
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Fonte del testo (vedi Tabella 6.3)

Codice di controllo 22: imposta il colore del testo

Potete sostituire il Codice di controllo 22 al primo argomento di un'istruzione COLOR. Cambiando il numero del colore con POKE INTIN, stabilirete il colore per il testo che comparirà in seguito sullo schermo.

Numero	Fonte di testo
1	fonte standard del sistema
2	svizzero
3	svizzero (sottile)
4	svizzero (corsivo sottile)
5	svizzero (chiaro)
6	svizzero (corsivo chiaro)
7	svizzero (corsivo)
8	svizzero (grassetto)
9	svizzero (corsivo grassetto)
10	svizzero (pesante)
11	svizzero (corsivo pesante)
12	svizzero (nero)
13	svizzero (corsivo nero)
14	olandese romano
15	olandese (corsivo)
16	olandese (grassetto)
17	olandese (corsivo grassetto)

Tabella 6.3: fonti di testo usate con il Codice di controllo 21.

Celle di memoria

CONTRL (0) = 22
CONTRL (1) = 0
CONTRL (3) = 1
INTIN (0) = Colore che scegliete per il testo (vedi Tabella 4.1)

Codice di controllo 106: cambia lo stile del testo

Il Codice di controllo 106 è una delle funzioni VDISYS che potete usare con istruzioni PRINT. Ci sono 64 diversi stili di testo disponibili e traducendo un numero binario in decimale, potete ottenere lo stile di vostra scelta con solo alcune linee di programma. Con il comando INTIN, dovrete trasformare i binari in decimali, come viene spiegato di seguito.

Celle di memoria

CONTRL (0) = 106
CONTRL (1) = 0
CONTRL (3) = 1
INTIN (0) = Specifica quale stile di testo volete usare (in decimali, tradotti dal binario)

Bit 0 = 1 per il testo in grassetto, 0 per il normale
Bit 1 = 1 per intensità chiara, 0 per normale
Bit 2 = 1 per testo obbliquo, 0 per normale
Bit 3 = 1 per il sottolineato, 0 per normale
Bit 4 = 1 per testo profilato, 0 per normale
Bit 5 = 1 per testo ombreggiato, 0 per normale

Per ottenere i numeri decimali, prima scrivete gli zeri e gli uni dello stile che volete in un ordine tale che il bit 5 si trovi a sinistra e il bit 0 a destra. Per esempio, supponiamo che vogliate il testo ombreggiato (bit 5 = 1), non profilato (bit 4 = 0), sottolineato (bit 3 = 1), non obliquo (bit 2 = 0), ad intensità chiara (bit 1 = 1) in grassetto (bit 0 = 1); il vostro numero binario sarà 101011, che si traduce nel decimale 43. POKE INTIN,43 programmerà questo stile di testo in memoria.

Esempio di programma Per dare un'occhiata ai 64 possibili diversi stili di testo, lanciate il seguente breve programma. La prima videata che ne risulta corrisponde alla Figura 6.15.

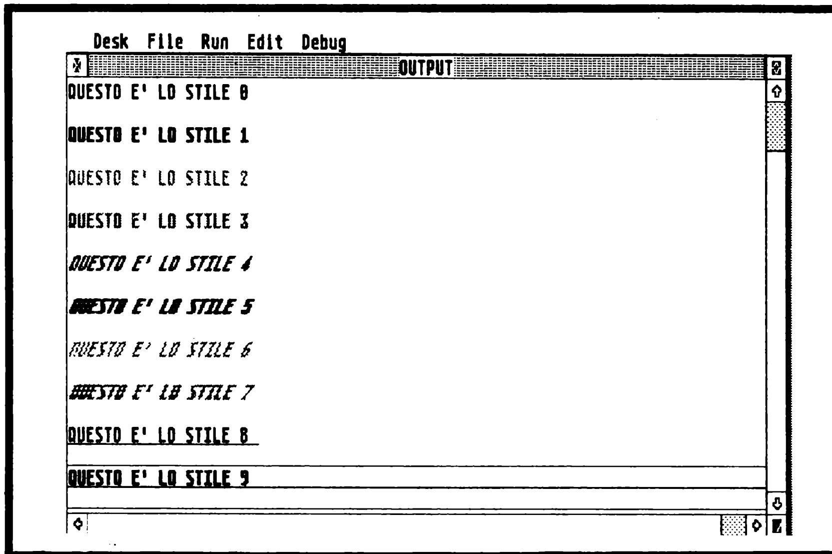


Figura 6.15: alcuni dei 64 stili di testo dell'ST.

```
  5 REM I 64 stili di testo dell'ST
  7 DEF SEG = 0
 10 FULLW 2: CLEARW 2
 20 POKE CONTRL+2,0
 30 POKE CONTRL,106
 40 POKE CONTRL+6,1
 50 FOR I=0 TO 63
 60 POKE INTIN,I
 70 VDISYS (1)
 80 PRINT "Questo è lo stile";I
 85 PRINT
 90 NEXT
100 GOTO 100
```

Codice di controllo 107: cambia l'altezza del testo

Per ottenere effetti interessanti, potete cambiare l'altezza del testo che state impiegando (sia il testo grafico che quello normale) con il Codice di controllo 107.

State attenti quando usate questo codice, perché tutto il testo (compresi i

comandi e le istruzioni che batterete per ritornare al testo normale) può risultare distorto oltre una certa altezza e diventare difficile da leggere.

Celle di memoria

CONTRL (0)=107
CONTRL (1)=0
CONTRL (3)=1
INTIN (0)=Altezza del carattere (in micron)

Esempio di programma Col prossimo programma si può provare il dimensionamento dell'altezza del testo e per tornare al testo normale si dovrà tornare al GEM per un momento e poi accedere di nuovo al BASIC ST. Il modo più veloce per farlo è di battere SYSTEM e premere Return.

```
 5 REM Cambiare l'altezza dei caratteri
 7 DEF SEG = 0
10 FULLW 2: CLEARW 2
20 POKE CONTRL,107
30 POKE CONTRL+2,0
40 POKE CONTRL+6,1
45 FOR H=1 TO 30
50 POKE INTIN,H
60 VDISYS (1)
70 PRINT "Salve; io sono alto";H;" micron!"
80 NEXT
90 POKE INTIN,12; VDISYS (1)
```

TESTO E GRAFICA VDISYS

La radice di tutte queste funzioni VDISYS è il Codice di controllo 1, che apre la stazione di lavoro virtuale GEM. Potete provare questa funzione per apportare dei cambiamenti necessari che altrimenti potrebbero richiedere diverse chiamate VDISYS.

Questa funzione ha una grande potenza, ma state attenti con certe operazioni (come cambiare la larghezza dello schermo), poiché testo e grafica potrebbero risultarne distorti oltre ogni limite di leggibilità se portate la larghezza dello schermo ad un valore insolito (come 13).

Codice di controllo 1: apre una stazione di lavoro

Celle di memoria

CONTRL (0)=1
CONTRL (1)=0
CONTRL (3)=11
INTIN (1)=Tipo di linea (vedi Codice di controllo 15)
INTIN (2)=Colore del polyline (vedi Tabella 4.1)
INTIN (3)=Tipo di polymarker (vedi Codice di controllo 18)
INTIN (4)=Colore di polymarker (vedi Tabella 4.1)
INTIN (5)=Fonte del testo (vedi Codice di controllo 21)
INTIN (6)=Colore del testo (vedi Tabella 4.1)
INTIN (7)=Stile interno per la routine di colorazione del fondo (vedi Codice di controllo 23)
INTIN (8)=Indice per la routine di colorazione del fondo (vedi Codice di controllo 24)
INTIN (9)=Colore per il motivo di colorazione del fondo (vedi Codice di controllo 25)

Celle di output

INTOUT (0)=Massima larghezza possibile dello schermo (in pixel)
INTOUT (1)=Altezza massima dello schermo (in pixel)
INTOUT (6)=Numero di tipi di linea disponibili
INTOUT (7)=Numero di larghezze di linea
INTOUT (8)=Numero di tipi di polymarker (fino a 6)
INTOUT (9)=Numero di dimensioni di polymarker
INTOUT (10)=Numero di stili tipografici disponibili
INTOUT (11)=Numero di modelli di stili tipografici
INTOUT (12)=Numero di stili di tratteggio
INTOUT (13)=Colori disponibili
INTOUT (15)=1 se la funzione rettangolo (GDP 1) è attivata, 0 se non lo è
INTOUT (16)=1 se la funzione arco è attivata (GDP 2), 0 se non lo è
INTOUT (17)=1 se il settore (GDP 3) è attivato, 0 se non lo è
INTOUT (18)=1 se il cerchio (GDP 4) è attivato, 0 se non lo è
INTOUT (19)=1 se l'ellisse (GDP 5) è attivato, 0 se non lo è
INTOUT (20)=1 se l'arco di ellisse (GDP 6) è attivato, 0 se non lo è
INTOUT (21)=1 se il settore di ellisse (GDP 7) è attivato, 0 se non lo è
INTOUT (22)=1 se il rettangolo arrotondato (GDP 8) è attivato, 0 se non lo è

INTOUT (23) = 1 se il rettangolo arrotondato pieno (GDP 9) è attivato, 0 se non lo è
INTOUT (24) = 1 per allineare il testo, 0 altrimenti
INTOUT (35) = 1 se potete produrre il colore sul monitor, 0 se non potete
INTOUT (39) = Numero di colori disponibili nella tavolozza (fino a 16)

OPERAZIONI GEMSYS

GEMSYS, come VDISYS, impiega una funzione contenuta nella memoria del computer. Non ci sono molte operazioni GEMSYS utili al programmatore in BASIC ST, ma per darvi un'idea di come funzionano, vediamo un breve esempio che impiega GEMSYS (79).

Il numero tra parentesi dopo GEMSYS è chiamato *codice dell'operazione* o *opcode* e potete accedere direttamente a qualsiasi funzione AES includendo il suo *opcode*. Per qualsiasi funzione GEMSYS, dovete prima includere queste linee nel vostro programma per impostare i necessari valori:

```
10 A# =GB
20 CONTROL = PEEK(A#)
30 GLOBAL = PEEK (A# + 4)
40 GINTIN = PEEK (A# + 8)
50 GINTOUT = PEEK (A# + 12)
60 ADDRIN = PEEK (A# + 16)
70 ADDROUT = PEEK (A# + 20)
```

La routine seguente controlla il mouse (proprio come il comando VDISYS chiamato Codice di Controllo 124). L'informazione per questa operazione GEMSYS è:

Codice = 79
GINTOUT (1) = Ascissa del puntatore del mouse
GINTOUT (2) = Ordinata del puntatore del mouse
GINTOUT (3) = Posizione dei bottoni del mouse
0 Nessuno premuto
1 Solo il sinistro
2 Solo il destro
3 Entrambi premuti

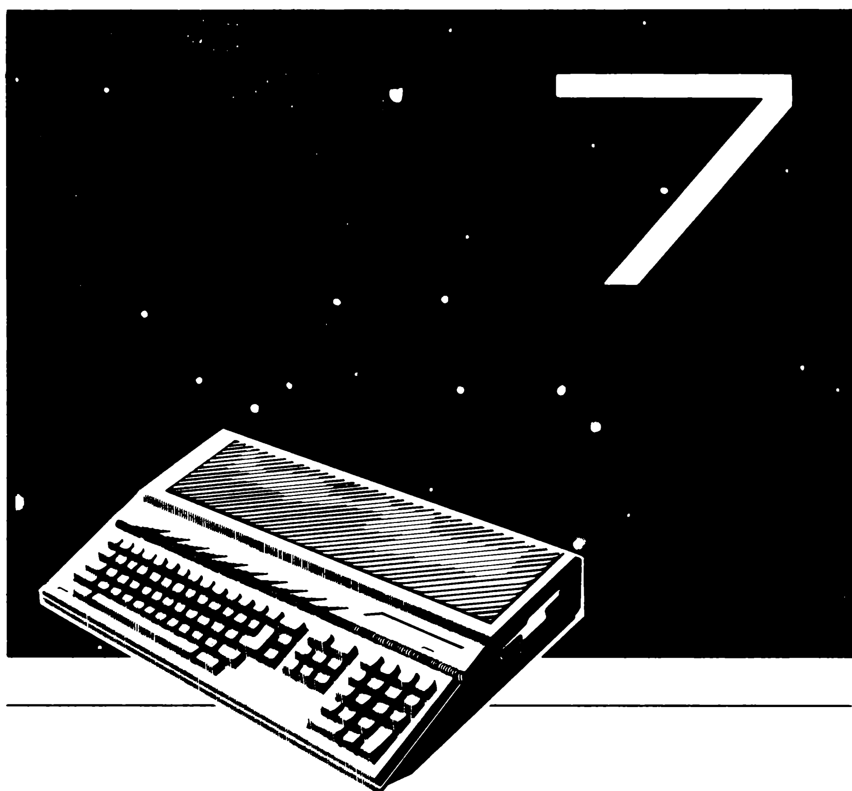
Con queste informazioni potete scrivere alcune altre linee per visualizzare informazioni immediate circa le coordinate del mouse e lo stato dei bottoni:

```
75 GEMSYS (29)
```

```
80 PRINT "Ascissa = ";PEEK (GINTOUT + 2);",Ordinata = ";PEEK
   (GINTOUT + 4);", e posizione del bottone = ";PEEK (GINTOUT + 6)
90 PRINT: GOTO 75
```

LA POTENZA DELLE FUNZIONI INCORPORATE

La grande varietà di operazioni VDISYS disponibili rende la programmazione molto più facile. Alcune di esse sono esclusive del VDISYS (quali il rettangolo pieno, che non potreste disegnare con nessun'altra singola parola chiave del BASIC ST) e tutte sono piuttosto veloci e facili da usare. Provate tutte le routine che vi possono interessare e provate a combinarle per creare programmi più grandi e complessi. Non ci vorrà molto perché questa caratteristica del vostro computer ST diventi indispensabile alla vostra programmazione.



COME USARE IL DRIVE E LA STAMPANTE

Fino a questo momento avete imparato in che modo ottenere quello che volete dal computer. Comunque, tutto quello di cui siete a conoscenza circa il BASIC ST fino a questo momento, non va oltre il computer; infatti avete imparato in che modo mandare od ottenere informazioni solo dallo schermo e dalla tastiera.

Come avrete ormai capito, il vostro ST può fare molto di più che non visualizzare grafici o emettere segnali acustici attraverso l'altoparlante. Come la maggior parte dei computer, può usare varie periferiche per ampliare la potenza del sistema nel suo complesso: per mandare e ricevere file con un modem, per comunicare con altri computer nella rete e per interagire con sintetizzatori musicali.

Comunque non c'è bisogno d'inoltrarsi tanto per adesso, dal momento che dovreste prima imparare a lavorare con gli strumenti che già avete a disposizione. La maggior parte di questo capitolo riguarderà l'uso del drive; vi verranno inoltre illustrati nuovi modi per mandare l'output alla vostra stampante (se ne possedete una) e per ottenere le informazioni dalla tastiera.

LE PAROLE CHIAVE D'INPUT E DI OUTPUT

L'*input* è l'insieme delle informazioni che il vostro computer riceve - dalla tastiera, dal mouse o dal modem - e l'*output* è l'insieme d'informazioni che il computer manda ad altri computer o dispositivi - al drive, alla stampante, allo schermo o a un modem.

La funzione INP

Una semplice parola chiave del BASIC ST che può ricevere un input è INP. Il suo formato è

INP *(porta)*

Questa parola chiave può ricevere un byte d'informazioni da una particolare *porta* e dirvi il valore di quel byte oppure se esiste o no a quella porta un byte da leggere. I numeri delle porte sono 0 per la stampante, 1 per l'interfaccia RS232 (il modem), 2 per la tastiera e 3 per il MIDI (che viene usato per interfacciare degli strumenti musicali col computer). Quando date un numero positivo come porta, il computer vi dirà il byte che si riceve da quella porta, ma non chiede all'utente nessun tipo d'input. Un numero negativo indica

che volete controllare lo stato di quella porta; se lo stato è 0, non ci sono caratteri disponibili da quella porta, mentre lo stato -1 indica che è presente un byte.

Il primo programma di questo capitolo è una semplice dimostrazione della funzione INP. Quando il computer incontra l'istruzione di assegnazione *X=INP (2)* alla linea 20, aspetterà che voi premiate un tasto qualsiasi. Una volta premuto un tasto, il computer controllerà che il byte ricevuto sia l'equivalente dei valori decimali 103 o 71 (i codici ASCII di g e G). Se il byte ricevuto non è l'equivalente di uno di questi due numeri, il computer vi chiederà allora di premere G. Se invece il test logico non è verificato (se G *era* davvero il tasto che avevate premuto), il computer vi ringrazierà per avere premuto G.

```
 5 REM Premete G quando il computer lo richiede
10 FULLW 2: CLEARW 2
15 PRINT "Premete G adesso."
20 X=INP (2)
30 IF X<>103 AND X<>71 THEN PRINT "Per favore premete G.":
   GOTO 20
40 PRINT "Grazie... ne avevo bisogno."
RUN
Premete G adesso.
T
Per favore premete G.
F
Per favore premete G.
G
Grazie... ne avevo bisogno.
```

Notate che INP opera con i codici ASCII, il che la rende una parola chiave utile per la programmazione. Per esempio, se avete un menù di scelte sullo schermo, il computer aspetta che voi premiate un tasto prima di procedere. Anche se INPUT funziona allo stesso modo di INP, quest'ultimo è un pò più elegante. Potreste usare INP anche nei giochi, in cui i giocatori video-logorati non vogliono premere un tasto e poi Return ogni volta che devono spostare la loro navetta spaziale.

L'istruzione WAIT

WAIT è un'istruzione simile a INP. Il suo formato è

WAIT *porta,numero*

che dice al computer di aspettare finché non viene ricevuto un certo valore *(numero)* dalla *porta* designata.

Se desiderate che il computer aspetti finché non premete la barra spaziatrice (codice ASCII 32), dovete battere *WAIT 2,32* (la porta 2 corrisponde alla tastiera). Comunque, fate attenzione quando usate WAIT, poiché il computer si fermerà finché il numero che WAIT sta aspettando non passa per la porta; se vi doveste bloccare mentre usate WAIT, dovrete resettare il computer.

L'istruzione OUT

L'istruzione OUT è l'opposto della funzione INP e il suo formato è

OUT *porta,byte*

che manda un *byte* ad una particolare *porta*. Il byte che mandate può andare da 0 a 65535 e i valori delle porte sono gli stessi usati per la funzione INP.

L'istruzione LPRINT

Una delle periferiche a cui è più facile mandare informazioni è la stampante. La stampante è un dispositivo molto utile, poiché rende possibile listare i programmi, stampare lettere e relazioni e creare interessanti stampe di grafici. Dire alla stampante di stampare qualcosa su carta è facile come dire al computer di stampare qualcosa sullo schermo, dal momento che i comandi LPRINT e LLIST funzionano allo stesso modo dei comandi PRINT e LIST con cui ormai avrete una certa familiarità.

Il formato di LPRINT è

LPRINT *espressione*

dove *espressione* può essere una variabile (N, G$, H(3), ILC), una stringa ("Salve", "**** TEST PER LA STAMPANTE ****"), un numero, un'equazione matematica o qualsiasi altra cosa che potreste visualizzare usando PRINT. Potete usare anche LPRINT USING esattamente allo stesso modo di PRINT USING.

Il prossimo programma impiega l'istruzione LPRINT per stampare delle informazioni su carta. Il programma comincia cancellando lo schermo e stampando 20 asterischi che formano il lato superiore della cornice:

```
10 REM Stampa una cornice
20 LPRINT STRING$ (20,"*")
```

Per completare la cornice, il programma usa il loop FOR...NEXT e stampa 10 linee composte ognuna da due asterischi; gli asterischi vengono stampati a 18 spazi di distanza in modo che formino i lati destro e sinistro della cornice (linea 30). Infine, l'istruzione LPRINT sulla linea 40 la completa disegnandone il lato inferiore:

```
30 FOR I=1 TO 10:LPRINT "*";SPACE$(18);"*":NEXT
40 LPRINT STRING$ (20,"*")
```

Il comando LLIST e la funzione LPOS (0)

Potreste listare alla stampante il precedente programma (o qualsiasi programma BASIC ST) battendo *LLIST.* LLIST funziona allo stesso modo di LIST e ha lo stesso formato, il che vuol dire che potete stampare con la stampante sia determinati gruppi di linee, sia una linea sola.

Inoltre, potete trovare la posizione della testina della stampante usando la funzione LPOS (0). *X=LPOS (0)* memorizza la posizione della testina in X e *PRINT LPOS (0)* la visualizza immediatamente. Quando la testina è all'estrema sinistra, la sua posizione è 0, il che significa che per ogni volta che si sposta verso destra dello spazio di un carattere, la sua posizione aumenta di 1.

L'istruzione WIDTH

Un'altra semplice istruzione di output è WIDTH, che cambia la lunghezza delle righe che il computer usa sullo schermo o sulla stampante. Il formato di questa istruzione è

WIDTH *espressione*

dove *espressione* è il numero massimo dei caratteri ammessi su ogni linea. Potreste anche usare il comando

WIDTH LPRINT

che interessa la stampante invece che lo schermo. Cambiando l'ampiezza dello schermo, non cambia la misura dei caratteri visualizzati. Si cambia sem-

plicemente il numero dei caratteri che possono stare su ogni linea. La normale ampiezza dello schermo è di 72 caratteri, per cui se battete

```
PRINT STRING$ (72,"!")
```

il computer riempie un'intera linea di punti esclamativi. Se invece aveste usato *WIDTH 35*, la stessa istruzione avrebbe fatto stampare due linee ognuna di 35 punti esclamativi; quando il computer raggiunge il trentacinquesimo carattere di ogni linea, salta alla linea seguente.

Il programma che segue mostra in che modo WIDTH agisce sulla lunghezza delle linee dello schermo. Per prima cosa, vengono stampati 40 simboli di dollaro e quando l'ampiezza dello schermo viene ridotta a 20, la stessa istruzione STRING$ produce due linee ognuna composta di 20 simboli di dollaro:

```
 5 REM Variazione di larghezza
10 WIDTH 40
20 PRINT STRING$(40,36)
30 WIDTH 20
40 PRINT STRING$(40,36)
50 PRINT "Vista la differenza?"
60 WIDTH 40
70 GOTO 70
```

Probabilmente non userete molti comandi d'input/output, dal momento che le applicazioni che li richiedono (per esempio la comunicazione con un modem) sono abbastanza avanzate da giustificare il costo (attualmente nell'ordina della 50mila lire) di un buon pacchetto software. D'altra parte, sarebbe buona norma imparare ad usare questi comandi tutte le volte che è opportuno farlo; la tastiera e la stampante sono componenti vitali in un computer e potrete sfruttarli meglio, diventando padroni delle parole chiave del BASIC ST.

I FILE AD ACCESSO DIRETTO E I FILE SEQUENZIALI

Esistono due tipi di file in un programma Atari ST: i file ad accesso diretto e i file sequenziali. Per fare un paragone, supponete di dover scegliere fra due tipi di mezzi magnetici: una cassetta per registratore o un dischetto. Per usare la cassetta, dovete cominciare dall'inizio e farla girare finché non raggiungete il file che state cercando. I programmi sul nastro sono in ordine e, per ottenere il programma numero 21, dovete prima far passare gli altri 20. Sul dischetto invece i programmi non sono in ordine. Quando richiedete un programma,

il dischetto fa la ricerca e trova rapidamente ciò di cui avete bisogno.

In *questo* paragone, la cassetta è il file *sequenziale* e il dischetto è il file *ad accesso diretto.* Un file sequenziale riporta tutto in modo ordinato e se volete qualcosa dovete prima passare attraverso tutto ciò che viene prima. Invece un file ad accesso diretto vi permette di raggiungere direttamente le informazioni che state cercando. Con l'ST avete entrambi i tipi di file e ognuno di essi offre dei vantaggi. Iniziamo a parlare dei file ad accesso diretto.

I file ad accesso diretto

I file ad accesso diretto memorizzano le informazioni in modo che voi possiate accedervi direttamente. Un file ad accesso diretto è costituito da diversi record che possono essere indirizzi postali, informazioni su un prodotto, numeri di telefono e così via. Questi record non sono registrati secondo un ordine che abbia senso per il programmatore, ma per il computer i file ad accesso diretto sono facili da usare perché qualsiasi record dell'intero file può essere portato in memoria semplicemente specificando il numero del record.

Il programma che vedete nel Listato 7.1 dimostra in che modo potete memorizzare i record in un file ad accesso diretto e richiamarli direttamente. Usan-

```
10  REM Programma degli indirizzi
20  FULLW 2: CLEARW 2
30  OPEN "R",#1,"Indirizzi",500
40  FIELD #1,30 AS FL$,30 AS SL$,40 AS TL$
50  INPUT "Aggiungi un indirizzo o ne prendi uno";AN$
60  IF AN$="Prendo" THEN GOTO 140
70  INPUT "Batti il numero iniziale ";NM
80  PRINT "Nome ";NM;"--->";:LINE INPUT NA$
90  LINE INPUT "Via--->";SA$
100  LINE INPUT "CAP, Citta'--->";CSZ$
110  LSET FL$=NA$: LSET SL$=SA$: LSET TL$=CSZ$
120  PUT #1,NM:  INPUT "Batti 1 per inserirne un altro o  2  per
     ottenere un nome";RP
130  IF RP=1 THEN NM=NM+1: GOTO 80
140  INPUT "Quale numero ";NU
150  GET #1,NU
160  PRINT FL$: PRINT SL$: PRINT TL$
170  GOTO 140
```

Listato 7.1: il programma degli indirizzi

do questo programma potete inserire il nome, l'indirizzo, la città, lo stato e il codice CAP di una persona, il che costituisce un record. Potreste avere un elenco di 50 o più persone e conoscendo il numero di record relativo ad ogni persona, potreste trovare il loro indirizzo completo premendo un solo tasto.

Le prime tre linee del programma definiscono la finalità (linea 10), fanno in modo che la finestra Output si dilati all'intero schermo e sia cancellata (linea 20) e poi aprono un file chiamato Indirizzi lungo 500 byte e identificato come file #1 (dal momento che è il primo e unico file aperto). La prima volta che viene fatto girare il programma, viene costituito un nuovo file; in seguito, il computer aprirà semplicemente il file esistente in modo da poter mandare e ricevere informazioni usando il file Indirizzi.

FIELD La linea 40 usa la parola chiave FIELD, che destina un certo numero di byte per variabili di stringa in un file ad accesso diretto (non sono permesse le variabili numeriche, dal momento che i file ad accesso diretto non le memorizzano). Il formato di questa istruzione è

FIELD *#numero, byte AS variabile di stringa[,byte AS variabile di stringa,...]*

dove *#numero* è il numero del file ad accesso diretto in cui dovrebbe essere riservato lo spazio, *byte* è il numero di byte che dovrebbe essere riservato alla variabile di stringa e *variabile* è ciò per cui state creando lo spazio. Dopo FIELD potete elencare quante variabili volete, basta che specifichiate quanti byte volete per ognuna. Alla linea 40 sono stati riservati 30 byte per FL$ (la prima linea dell'indirizzo), 30 byte per SL$ (la seconda linea) e 40 byte per TL$ (la terza linea). Il numero del file è #1 dal momento che è l'unico file che stiamo usando. Ricordate che FIELD riserva lo spazio per le variabili in ogni *record* di un file, non per l'intero file. Quindi, dal momento che sono stati riservati 30 byte per FL$, potete disporre di un massimo di 30 caratteri per la prima riga di ogni record del file. Questo riserva lo spazio, ma FIELD non memorizza effettivamente le variabili.

Potete dire al computer se volete aggiungere un indirizzo al file o recuperare un indirizzo già memorizzato su disco. Se la risposta è uguale alla stringa Prendo, il programma andrà alla routine della linea 140 per recuperare un record in base al suo numero.

Se la vostra risposta è diversa da "Prendo", il programma va alla linea 70 per scoprire da quale numero deve cominciare a numerare i nuovi record. Se questa è la prima volta che usate il programma, il numero di partenza dovrebbe essere 1. Altrimenti, dovete battere un numero uguale al numero dell'ultimo record più 1; per esempio, se nel file ci sono già 43 indirizzi, il numero di partenza dovrebbe essere 44.

Adesso il computer può cominciare a ricevere informazioni relative a nome, via, città/stato/CAP-NA$, SA$ e CSZ$.

LSET Per convertire queste stringhe in un formato adatto alla memorizzazione in file, avete bisogno dell'istruzione LSET. Il suo formato è

LSET *nuova stringa=vecchia stringa*

dove *nuova stringa* è semplicemente il nome della nuova stringa che usate in modo che LSET possa convertire per il file la stringa che avete in memoria, e *vecchia stringa* è ciò che volete sia preparato per il file. Per esempio, se voleste memorizzare una stringa chiamata YY$, potreste battere *LSET H$=YY$*. Il nome H$ verrebbe memorizzato nel file come nome della stringa. Alla linea 110, le variabili FL$, SA$ e TL$ vengono usate nel file per rappresentare le variabili delle stringhe che avete appena battuto: NA$, SA$ e CSZ$.

PUT Adesso che tutte le stringhe sono state convertite e sono pronte per essere mandate al file, per completare l'operazione dovete solo usare l'istruzione PUT. Il formato di PUT è

PUT *#numero[,numero di record]*

dove *#numero* è il numero del file (#1) e il numero di record è uguale alla variabile (NM). La linea 120 usa PUT #1,NM per trasferire le variabili FL$, SL$ e TL$ al file sotto i loro nomi rispettivi, quindi il computer vi chiede se desiderate inserire un'altra registrazione o spostarvi alla funzione di ricerca.

Se RP è uguale a 1, per indicare che volete inserire un altro indirizzo, il programma aumenta di 1 il numero in uso di record (la variabile NM) e ritorna alla linea 80 in modo che possiate inserire un altro record. Se la variabile RP non è uguale a 1, il programma va alla linea 140, dove è situata la routine di ricerca. Alla linea 140 potete inserire il numero di registrazione dell'indirizzo che volete vedere.

GET Una volta che il computer ha il numero di registrazione, può memorizzarlo con l'istruzione GET, il cui formato è lo stesso di PUT

GET *#numero[,numero di record]*

GET ricerca il numero di record NU dal file #1 e una volta che il record è in memoria, il programma visualizza il nome (FL$), la via (SL$) e città/stato/CAP (TL$) che corrispondono a quel numero di registrazione. Il GOTO 140 alla linea 170 fa in modo che il programma ritorni all'inizio della routine

di recupero in modo che sullo schermo compaia un altro record. Per uscire dal programma, premete Control-G.

Output Ecco un esempio di output usando questo programma:

RUN
Aggiungi un indirizzo o ne prendi uno? Aggiungo
Batti il numero iniziale? 1
Nome 1---> Alfredo Rossi
Via---> Altopascio 20
CAP, Città---> 20121 Milano
Batti 1 per inserirne un altro o 2 per ottenere un nome? 1
Nome 2---> Roberto Bianchi
Via---> Saltamerenda 12
CAP, Città---> 00196 Roma
Batti 1 per inserirne un altro o 2 per ottenere un nome? 1
Nome 3---> Alberto Francavilla
Via ---> Raffaello 57
CAP, Città---> 21100 Varese
Batti 1 per inserirne un altro o 2 per ottenere un nome? 2
Quale numero? 2
Roberto Bianchi
Saltamerenda 12
00196 Roma
Quale numero?

Un metodo più diretto

Il vantaggio dei file ad accesso diretto è che vi permettono di trovare delle informazioni in modo più veloce che non con i file sequenziali. Per evitare di dovervi ricordare a quali numeri corrispondono certi nomi, potete modificare il programma precedente, come mostrato dal Listato 7.2. La maggior parte del programma rimane invariato, ma metterò in evidenza i cambiamenti apportati perché possiate trovare un indirizzo semplicemente battendo il nome della persona.

Per prima cosa, viene data l'istruzione alla linea 10 in caso vogliate inserire un nome che non è ancora memorizzato in un file. Se il computer incontra un errore mentre fate girare il programma, andrà alla linea 200 e vi dirà il numero dell'errore in modo che il programma possa essere fermato; probabilmente riceverete un messaggio d'errore solo se battete un nome che non è ancora memorizzato in nessun record o che è stato battuto in modo leggermente diverso da come l'avevate specificato.

```
10  ON ERROR GOTO 200
20  REM programma degli indirizzi modificato
30  FULLW 2: CLEARW 2
40  OPEN "R",#1,"Indirizzi",500
50  FIELD #1,30 AS FL$,30 AS SL$,60 AS TL$
60  INPUT "Aggiungi un indirizzo o ne prendi uno";AN$
70  IF AN$="Prendo" THEN GOTO 150
80  INPUT "Batti il numero iniziale";NM
90  PRINT "Nome ";NM;"--->";:LINE INPUT NA$
100  LINE INPUT "Via--->";SA$
110  LINE INPUT "CAP, Citta'--->";CSZ$
120  LSET FL$=NA$: LSET SL$=SA$: LSET TL$=CSZ$
130  PUT #1,NM:  INPUT "Batti 1 per inserirne un altro o 2 per
     ottenere un nome";RP
140  IF RP=1 THEN NM=NM+1: GOTO 80
150  INPUT "Batti il nome di una persona";NS$
160  NU=1
170  GET #1,NU
180  IF  LEFT$(FL$,LEN(NS$))=NS$  THEN  PRINT  SL$:  PRINT  TL$:
     PRINT"#"NU: ?: GOTO 130
190  NU=NU+1: GOTO 170
200  PRINT "Errore --numero "ERR: END
```

Listato 7.2: programma degli indirizzi modificato

La linea 150 chiede il nome della persona di cui volete l'indirizzo. Una volta che avete battuto il nome, il computer comincia a scorrere tutti i numeri di record con l'istruzione GET della linea 170. Quindi, invece di battere *IF FL$=NS$* nella linea seguente per vedere se si era verificata un'uguaglianza fra NS$ e la prima linea del record, ho dovuto usare anche l'istruzione LEFT$. La ragione è che per ogni variabile che memorizzate in un file, il computer userà il numero di byte che avete specificato per quella variabile e riempirà il resto dello spazio con degli spazi vuoti. Alla variabile FL$ sono stati dati dall'istruzione FIELD 30 byte per ogni record. Poiché la maggior parte dei nomi sono composti da meno di 30 caratteri, i rimanenti byte erano vuoti. Per cui, anche se NS$ e FL$ erano uguali, non coinciderebbero, dal momento che FL$ conterrebbe dei caratteri in più (vuoti) rispetto a NS$. Per risolvere questo problema, alla linea 180 ho usato il test logico:

```
IF LEFT$(FL$,LEN(NS$))=NS$
```

Il computer controlla solo i caratteri all'estrema sinistra, basandosi sulla lunghezza della stringa che avete battuto come nome (NS$). Se il nome che avete inserito è lungo 15 caratteri, il programma controlla i primi 15 caratteri di FL$ che trova in ogni record. Se si verifica la corrispondenza, il resto dell'indirizzo di quel record (SL$ e TL$) verranno stampati insieme al numero del record in cui è stato trovato l'indirizzo. Come potete vedere nella linea 180,

una volta che l'indirizzo è stato visualizzato, il programma ritorna alla linea 130 per vedere se volete trovare un altro nome o se volete inserire un altro indirizzo. Finché non riscontra una corrispondenza, tuttavia, il computer continua ad incrementare NU di uno e a ritornare alla linea 170 finché non trova il record il cui campo FL$ corrisponda al nome inserito come NS$. Se il file è completamente esaurito e non si verifica una corrispondenza, il programma andrà alla routine di errore alla linea 200 per mostrare il numero dell'errore che si è verificato e chiuderà il programma.

Ecco un esempio di output che usa questo programma:

RUN
Aggiungi un indirizzo o ne prendi uno? Aggiungo
Batti il numero iniziale? 1
Nome 1---> Alfredo Rossi
Via---> Altopascio 20
CAP, Città---> 20121 Milano
Batti 1 per inserirne un altro o 2 per ottenere un nome? 1
Nome 2---> Roberto Bianchi
Via---> Saltamerenda 12
CAP, Città---> 00196 Roma
Batti 1 per inserirne un altro o 2 per ottenere un nome? 1
Nome 3---> Alberto Francavilla
Via ---> Raffaello 57
CAP, Città---> 21100 Varese
Batti 1 per inserirne un altro o 2 per ottenere un nome? 2
Batti il nome di una persona? Roberto Bianchi
Saltamerenda 12
00196 Roma
Batti 1 per inserirne un altro o 2 per ottenere un nome?

LSET e RSET Ci sono alcune altre parole chiave che dovete conoscere per usare i file ad accesso diretto. Avete già una certa familiarità con LSET, che porta le stringhe in memoria e le prepara perché vengano memorizzate in un file ad accesso diretto. Se una stringa è più corta dei byte ad essa riservati, LSET riempirà i byte che rimangono alla destra della stringa con degli spazi vuoti. Se, per esempio avete a disposizione 10 byte per AS$ e AS$ ha solo 8 caratteri, gli ultimi due byte verranno memorizzati come spazi vuoti. Usare RSET invece di LSET farà in modo che il computer aggiunga degli spazi vuoti per riempire lo spazio che precede la stringa (anziché quello che la segue). Se A$ fosse "********", LSET lo memorizzerebbe come "******** " (due spazi vuoti alla fine) e RSET lo registrerebbe nel record come " ********"

(due spazi vuoti all'inizio). Questa non è una parola chiave importante, ma in alcune occasioni potreste trovarla di qualche utilità.

**MKD$, MKS$ e MKI$** Le funzioni MKD$, MKS$ e MKI$ sono piuttosto importanti, poiché convertono i numeri in stringhe che possono essere memorizzate. I file ad accesso diretto non accettano variabili numeriche, quindi per convertire un numero o una variabile numerica in una stringa adatta per l'uso di LSET, esistono tre formati. Per i numeri a precisione doppia, userete

LSET *stringa* =MKD$*(espressione numerica)*

Per i numeri a precisione singola, userete

LSET *stringa* = MKS$*(espressione numerica)*

Per i numeri interi, userete

LSET *stringa* = MKI$*(espressione numerica)*

Come potete aspettarvi dai rispettivi tipi di numero, le lunghezze di queste stringhe saranno di otto byte, quattro byte e due byte rispettivamente. Una volta che avete salvato queste informazioni in un file, avrete bisogno di recuperarle e di riconvertirle in numeri. I comandi per fare questo sono CVD, CVS e CVI. Supponete che le stringhe che avete usato per i numeri a precisione doppia e a precisione singola e per i numeri interi fossero DB$, SN$ e IN$, e che voleste riportare questi numeri in memoria e visualizzarli. Prima caricate le stringhe con GET, poi battete

```
PRINT CVD (DB$)
PRINT CVS (SN$)
PRINT CVI (IN$)
```

È chiaro quindi che i file ad accesso diretto sono facili da usare. Una volta che avete imparato ad usare le parole chiave per salvare e caricare le stringhe usando il drive, sarete in grado di realizzare delle applicazioni utili.

I file sequenziali

Quando usate i file sequenziali, dovete ricordarvi che i record sono memorizzati in un ordine fisso. Per ottenere il record 9, dovete passare attraverso

```
10 REM Programma con file sequenziale
20 REM Vi permette di fare l'input delle vendite per i giorni di
   un mese
30 FULLW 2: CLEARW 2
40 DIM SD$(31): INPUT "Batti 1 per inserire i dati, 2 per
   caricarli";DE
50 IF DE=2 THEN GOTO 110
60 INPUT "Di che mese si tratta ";MN$
70 OPEN "O",#1,MN$
80 INPUT "Quanti giorni di vendita ci sono in questo mese ";SD:
   PRINT #1,SD
90 FOR I=1 TO SD: PRINT "Vendite del giorno ";I;: INPUT SD$(I)
100 PRINT #1,SD$(I): NEXT: GOTO 150
110 INPUT "Che mese vuoi vedere ";MN$
120 OPEN "I",#1,MN$: INPUT #1,SD
130 FOR I=1 TO SD: INPUT #1,SD$(I): PRINT "Le vendite del giorno "
    ;I;" erano Lit.";SD$(I)
140 NEXT I
150 CLOSE #1
160 ERASE SD$: GOTO 40
```

Listato 7.3: programma con file sequenziale

i record da 1 a 8. Il programma mostrato nel Listato 7.3 vi permette di inserire le cifre delle vendite per alcuni giorni del mese. Le linee 10-30 definiscono lo scopo e preparano la finestra di Output. Nella linea 40, il programma costruisce una matrice chiamata SD$() per memorizzare le cifre relative alle vendite per 31 giorni al mese. Il programma poi chiede se volete inserire dei dati o volete caricarli. Se il numero che viene battuto è il 2 (linea 50), il programma va alla linea 110 in modo che possa caricare dei dati.

Se volete inserire dei dati, il computer chiede il nome del mese a cui si riferiscono i dati di vendita (linea 60). L'istruzione OPEN alla linea 70 ha un formato diverso da quello usato per i file ad accesso diretto (il suo formato verrà descritto più avanti). La "O" indica che il programma manderà dei dati ad un file sequenziale, il #1 dà un numero al file e MN$ gli dà il nome (che è il nome del mese).

PRINT Il computer ha bisogno di sapere quanti giorni di vendita ci sono in un mese in modo che possa far girare un loop FOR...NEXT per avere tutti i dati. Una volta che è stato fatto l'input relativo al numero dei giorni, il computer registra quel numero nel file con il comando PRINT #1,SD (linea 80). PRINT# è l'istruzione più utile per mandare le informazioni ad un record e il suo formato è

PRINT *#numero, voce[, voce,...]*

dove *voce* può essere un numero, delle stringhe, delle variabili numeriche o qualsiasi altra cosa che potete usare con l'istruzione PRINT. Se specificate più di una voce, ricordate di separarle con la virgola o con il punto e virgola.

Poi il computer comincia il loop che conta da 1 a SD (linea 90). Inserite le cifre delle vendite per ciascun giorno e il programma registra questa stringa su disco sotto il suo unico nome di matrice. Una volta che il loop finisce (linea 100), il computer va alla linea 150 per chiudere il file, cancella il contenuto della matrice e ritorna alla linea 40 per vedere se volete inserire ulteriori dati o se volete caricarne.

INPUT Se decidete di caricare dati, il computer vi darà il nome del mese che avete chiesto di vedere. Con quella informazione, il computer può aprire il file e scoprire quanti giorni sono memorizzati con il comando INPUT #1, SD. L'istruzione INPUT# funziona quasi allo stesso modo di INPUT che già conoscete, eccetto che il computer carica delle informazioni che sono già memorizzate nel file. Il formato di INPUT# è

INPUT *#numero,variabile[,variabile,...]*

Con INPUT# potete caricare una o più variabili e il computer assegnerà i nomi di variabile che avete specificato alle voci che vengono caricate dal disco. Con *INPUT #1,SD* alla linea 120, il computer carica il numero che indica quanti giorni di vendita ci sono in un mese e memorizza quel numero nella variabile SD.

Alla linea 130, il programma comincia il loop FOR ... NEXT che conta da 1 a SD. Man mano che le cifre di ogni giorno vengono caricate nella matrice SD$(), vengono visualizzate sullo schermo finché non sono stati caricati tutti i record di vendita dal file.

Output Per dimostrare in che modo possa funzionare questo programma, ecco un'esecuzione campione usando solo cinque giorni del mese di giugno:

RUN
Batti 1 per inserire i dati, 2 per caricarli? 1
Di che mese si tratta? Giugno
Quanti giorni di vendita ci sono in questo mese? 5
Vendite del giorno 1? 1507
Vendite del giorno 2? 1223
Vendite del giorno 3? 2553
Vendite del giorno 4? 927
Vendite del giorno 5? 2012
Batti 1 per inserire i dati, 2 per caricarli? 2

Che mese vuoi vedere? Giugno
Le vendite del giorno 1 erano Lit. 1507
Le vendite del giorno 2 erano Lit. 1223
Le vendite del giorno 3 erano Lit. 2553
Le vendite del giorno 4 erano Lit. 927
Le vendite del giorno 5 erano Lit. 2012

LINE INPUT Alcune altre istruzioni possono risultare utili quando scrivete dei programmi con i file sequenziali. LINE INPUT # funziona quasi allo stesso modo di INPUT #, ma può caricare anche le virgole, i punti di domanda e altri caratteri che non sarebbero caricati con INPUT #. Il suo formato è

LINE INPUT *#numero, variabile di stringa*

dove, come al solito, *#numero* è il numero del file e la *variabile di stringa*, caricata dal disco, può essere lunga fino a 254 caratteri. Per capire la differenza fra LINE INPUT # e INPUT #, rileggete le sezioni su LINE INPUT e INPUT nel Capitolo 2; le differenze fra i due comandi sono circa le stesse per entrambe le coppie di comandi, eccetto per il fatto che quelli normali operano con la tastiera anziché con il drive.

WRITE e WRITE # Un'istruzione di output che potete usare sia con lo schermo che con il drive è WRITE. WRITE (o WRITE # se mandate le informazioni al drive) funziona quasi allo stesso modo dell'istruzione PRINT, eccetto per il fatto che mette le virgolette ad ogni voce che viene stampata. Questo vi aiuta a distinguere i singoli dati che vengono stampati sullo schermo. WRITE è utile in modo particolare poiché vi mostra quali dati vengono mandati al file. Supponete di volere registare su un file sequenziale le variabili A$, B$ e C$. Il formato di WRITE è

WRITE *variabile o stringa di dati[,variabile o stringa di dati,...]*

Potreste registrare queste informazioni e vedere cosa è stato registrato includendo quanto segue nel vostro programma:

```
WRITE A$,B$,C$:WRITE #1, A$, B$, C$
```

La prima istruzione WRITE visualizza le tre stringhe sullo schermo e l'istruzione WRITE # manda le stringhe al file #1 che al momento è aperto. Sia sullo schermo che sul file, ognuna delle tre stringhe è chiusa fra virgolette.

EOF Infine, la funzione EOF vi permetterà di sapere quando avete raggiunto la fine di un file sequenziale. Nei vostri programmi su file sequenziali dovreste includere una linea che controlli lo status del file. Il formato di EOF è

EOF *(numero del file)*

Supponete di avere aperto il file 3; finché EOF (3) è uguale a 0, nel file restano ulteriori informazioni da leggere. Tuttavia, una volta che è stato letto l'ultimo dato, EOF darà il valore di —1. Usando EOF nei vostri programmi di file sequenziali, potete evitare di andare oltre la fine di un file e quindi di far fermare il programma per condizione anomala.

LE FUNZIONI DEL DISCO

Avete già una certa familiarità con i comandi LOAD, SAVE e REPLACE che vengono usati con il drive: LOAD carica un programma dal disco, SAVE copia il programma dalla memoria e lo memorizza su disco e REPLACE salva nuovamente il programma in memoria con un nome già presente sul disco. Ognuno di questi comandi è seguito dal nome del file (fra virgolette) che volete caricare o salvare. Il BASIC ST ha molti altri comandi per il disco, alcuni dei quali sono importanti per avere dei file ben organizzati.

Quando state facendo una programmazione in BASIC la maggior parte dei file che salvate su disco, saranno programmi che avete scritto voi. Comunque il termine *file* può descrivere una vasta gamma di tipi di file che possono essere usati con il BASIC ST.

Un file, in termini normali, è un insieme d'informazioni che vengono memorizzate su disco. Ad ogni file che viene salvato va dato un nome unico e sarà utile che il nome sia allusivo al suo contenuto. Per rendere facile la definizione dei file, il BASIC ST vi permette di aggiungere un'*estensione* di tre lettere alla fine del nome del file. Per i programmi BASIC, la vostra estensione sarà BAS. Per i file di testo, potreste memorizzare il file con l'estensione TXT. I file di dati potrebbero essere identificati con DAT. Un punto separa il nome del file dalla estensione, per cui se aveste un file contenente le informazioni circa le vendite nel mese di aprile, potreste chiamare il file APRVEND.DAT. Notate che il nome del file può contenere fino a otto caratteri e che l'estensione non può essere più lunga di tre caratteri.

Per poter lavorare con i file in modo semplice, ricordate di dare dei nomi comprensibili. Non ammassate troppe informazioni nel nome di un file; per esempio, se doveste registrare un file di testo che descrive le prestazioni di James Smith durante il 1986, il nome del file JSPRFM86.TXT alcuni mesi dopo risulterebbe sicuramente incomprensibile. Un'alternativa migliore potrebbe es-

sere JSMITH86.PER. Notate che nell'estensione del nome potete mettere qualsiasi cosa voi vogliate. Il punto è di raggruppare dei file fra loro collegati con la stessa estensione, in modo che le vostre informazioni possano essere ben organizzate. Per esempio, potreste avere 20 file con l'estensione .PER ad indicare le prestazioni di 20 impiegati.

DIR

Un importante comando di disco è DIR, che visualizza i nomi dei file memorizzati su disco. DIR è l'abbreviazione di directory e il suo formato è

DIR *[drive:stringa di ricerca]*

Se battete solo DIR, il computer elencherà tutti i file che si trovano sul disco che state usando in quel momento. DIR seguito dalla designazione di un drive (o A: o B:) mostrerà i file che si trovano sul disco in quel drive; *DIR B:*, per esempio, visualizzerà i nomi dei file contenuti sul disco nel drive B.

Per cercare un particolare tipo di file, potete usare una *stringa di ricerca* che il computer usa per confrontare i nomi dei file. Il modo più semplice di usare questa caratteristica è quello di battere *DIR* seguito dal nome di un programma; se il programma si trova su un disco, verrà visualizzato il suo nome. Se non c'è, non verrà visualizzato niente. Per esempio, per vedere se il programma FUNGAME.BAS è memorizzato nel drive A, battete

DIR A: FUNGAME.BAS

I caratteri jolly

Un modo più preciso di cercare i file sono i *caratteri jolly* ? e *. Il punto di domanda si sostituisce a qualsiasi carattere e l'asterisco sostituisce qualsiasi carattere o gruppo di caratteri. Potete cercare dei nomi di programmi o delle estensioni ignorando il resto del nome del file. Ecco alcuni esempi di come potete usare DIR con una stringa di ricerca:

DIR GAME?.COM

visualizza ogni nome di file che comincia con GAME e ha per estensione COM. I nomi che coincidono con questa stringa potranno essre GAME1.COM, GAMEW.COM e GAME3.COM. Ricordate che il punto di domanda sosti-

tuisce un solo carattere, per cui GAMEWOW.COM non combacerà dal momento che la stringa di ricerca *GAME* è seguita da tre caratteri.

DIR B: *.BAS

elencherà tutti i file nel drive B con estensione BAS. L'asterisco accetta qualsiasi carattere o gruppi di caratteri, il che significa che MAC.BAS, STUTILITY.BAS e NEWCUBER.BAS combaceranno tutti con la stringa di ricerca.

DIR ???.*

I tre punti di domanda sostituiscono un nome di tre lettere, mentre l'asterisco sostituisce un'estensione qualsiasi. Per esempio, WOW.COM, VR3.EXE e DSL.BAS sono tre nomi che combaceranno con la stringa di ricerca, mentre PROGRAM.COM, FILENAME,APL e O1 non verranno visualizzati perché non sono lunghi tre caratteri.

L'istruzione NAME

Se volete cambiare il nome di un file che è già salvato su disco, potete usare l'istruzione NAME. Il suo formato è

NAME *nome vecchio AS nome nuovo*

dove *nome vecchio* è il nome attuale del file e *nome nuovo* è il nome nuovo che volete dare al file. Per cambiare il nome di un programma chiamato ROBOT.EXE in VALLEY.EXE, battete

NAME ROBOT.EXE AS VALLEY.EXE

Il comando ERA e l'istruzione KILL

Potete anche cancellare un file che non usate più con due parole chiave. ERA può essere usato solo nella modalità immediata; KILL può anche essere incluso nei programmi. I loro formati sono

ERA *[drive:] nome del file*
KILL *"stringa" o espressione di stringa*

Se voleste liberarvi di un file chiamato EATSPACE.BAS con ERA, dovreste battere *ERA EATSPACE.BAS* (o se il file si trovasse nel drive B e steste usando il drive A, *ERA B: EATSPACE.BAS).* Per ottenere lo stesso risultato con KILL, battete uno dei due comandi che seguono:

```
KILL "EATSPACE.BAS"
N$="EATSPACE.BAS": KILL N$
```

L'istruzione MERGE

Se voleste unire due programmi per ottenerne uno più grande, dovreste usare l'istruzione MERGE. Il suo formato è

MERGE *nome del file*

MERGE carica il file col nome che avete specificato e lascia il programma attualmente in memoria inalterato.

Inoltre, quando effettuate l'operazione MERGE qualsiasi variabile in memoria non viene cancellata. Ci sono alcune cose che dovete tenere presente quando volete unire dei programmi:

- Assicuratevi che i numeri delle linee dei programmi siano differenti. Se avete due programmi, entrambi contenenti le linee 10, 20 e 30, il programma che caricate sovrapporrà quelle linee alle corrispondenti nel programma che avete in memoria.
  È importante avere numeri di linea diversi in quanto eviterete di perdere delle linee importanti del programma in memoria e vi assicurerete anche che le linee vengano eseguite nell'ordine esatto.

- Per essere sicuri di non avere problemi con le linee dei programmi, usate l'istruzione RENUM con tutti i programmi che volete unire in modo che i numeri delle linee siano corretti. Per esempio, supponete di avere su disco questi due brevi programmi:

  Programma Uno

```
10 REM Questo è l'inizio del programma
20 PRINT "Salve, sono il computer Atari ST"
30 PRINT "Vorrei contare fino a 10 per te."
```

Programma Due

```
30 FOR X=1 TO 10: PRINT X
40 NEXT
50 PRINT "Fatto. Ciao!":END
```

Se usaste MERGE per unire questi due programmi, la linea 30 del programma uno verrebbe cancellata. Per risolvere questo problema, caricate il programma due e battete

```
RENUM 40
```

che farà in modo che le linee 30, 40 e 50 diventino le linee 40, 50 e 60. A questo punto, potreste unire senza problemi il programma uno ed ottenere un programma completo.

- È una buona idea salvare il programma che avete in memoria prima di unirlo ad un altro programma. Questo vi salvaguarderà da eventuali complicazioni durante l'operazione di unificazione, perché avrete sempre gli originali salvati su disco.

Segue ora un esempio di come potete usare MERGE con i programmi. Per prima cosa, battete il programma che segue e salvatelo con il nome PRIMO.BAS:

```
10 REM Primo.bas
20 FULLW 2: CLEARW 2
30 COLOR 1,3,2
40 FOR I = 3 TO 30 STEP 3
```

Una volta che avete salvato il programma, battete NEW per liberare la memoria del computer e battete le linee che seguono come seconda parte del programma:

```
50 REM Routine cerchio
60 PCIRCLE 150,75,1
70 COLOR 1,RND*16
80 NEXT
```

Adesso, usate l'istruzione MERGE e listate il programma unificato:

```
MERGE "PRIMO.BAS."
LIST
10 REM Primo.bas
20 FULLW 2: CLEARW 2
30 COLOR 1,3,2
40 FOR I=3 TO 30 STEP 3
50 REM Routine cerchio
60 PCIRCLE 150,75,1
70 COLOR 1,RND*16
80 NEXT
```

MERGE è utile quando avete delle routine preimpostate che usate spesso e che volete salvare su disco e unirle ad altri programmi sviluppati in un secondo momento. Potreste scrivere un'intera biblioteca di subroutine da unire ad altri programmi da voi sviluppati.

L'istruzione CHAIN

Una parola chiave simile a MERGE è CHAIN che non solo fonde più programmi ma conserva anche le variabili usate con il programma corrente; ha due formati. Il primo formato è

CHAIN *"nome del file"[,numero della linea,ALL]*

In questo formato, CHAIN carica un programma dal disco e lo fa girare. Se specificate un *numero di linea*, il *nome del file* del programma caricato comincerà a girare da quel numero di linea. Se non specificate il numero di linea, il programma comincia dall'inizio. Se volete conservare tutte le variabili in memoria, quando il nuovo programma comincia a girare, battete *ALL* dopo la virgola, come mostrato nella linea del formato. Ecco alcuni esempi del primo tipo di formato:

```
CHAIN "FUNGAME.BAS"
```

fa girare un programma chiamato FUNGAME.BAS. Potreste ottenere lo stesso risultato battento *RUN FUNGAME.BAS.* Però quando il programma comincia, tutte le variabili che si trovano in memoria vengono cancellate.

```
CHAIN "FUNGAME.BAS",50
```

fa girare un programma chiamato FUNGAME.BAS cominciando dalla linea 50. Quando il programma comincia tutte le variabili in memoria vengono cancellate.

```
CHAIN "FUNGAME.BAS",ALL
```

fa girare il programma FUNGAME.BAS dall'inizio (poiché non è stato specificato il numero della linea), ma tutte le variabili in memoria non vengono cancellate quando il programma comincia a girare.

Il secondo formato di CHAIN è il seguente:

CHAIN MERGE *"nome del file",linea*,DELETE *numero di linea[-numero di linea]*

Come nel precedente formato, tutte le variabili vengono conservate (il che elimina il bisogno di ALL), ma il programma che si trova attualmente in memoria non viene cancellato quando viene caricato l'altro.

Potete anche rimuovere certe linee dal programma in memoria prima che cominci l'operazione MERGE battendo *DELETE e* i numeri di linea che volete rimuovere alla fine dell'istruzione. Per cancellare le linee 50-70 di un programma, unire un file chiamato SUBRTINE.BAS con il programma attualmente in memoria e cominciare a far girare il programma alla linea 30, dovreste battere

```
CHAIN MERGE "SUBRTINE.BAS",30,DELETE 50-70
```

Ecco un esempio di CHAIN, usando il suo primo formato. Battete la linea 10 e salvatela con il nome CHAINER:

```
 5 REM Chainer
10 PRINT "Il numero inserito era";NUM
```

Adesso battete *NEW* per cancellare ogni cosa dalla memoria, inserite la linea che segue e battete *RUN* per far girare il programma:

```
10 FULLW 2: CLEARW2
20 INPUT "Inserisci un numero ";NUM
30 CHAIN "CHAINER",10,ALL
RUN
Inserisci un numero? 30
Il numero inserito era 30
```

L'istruzione COMMON

Se volete specificare le variabili da conservare quando date l'istruzione CHAIN, potete usare COMMON. Il suo formato è

COMMON *variabile[,variabile,...]*

dove *variabile* è qualsiasi variabile che volete salvare. Se battete

```
COMMON A$,PX,AR()
```

verranno conservate le variabili A$ e PX e la matrice AR(); quando il secondo programma nell'esempio precedente è stato messo in memoria, ogni altra variabile all'infuori di quelle verrà cancellata.

PROGRAMMAZIONE CON I FILE

Una delle cose più utili da imparare a fare con il BASIC ST è di lavorare con le informazioni dal drive. Il disco è il mezzo dell'Atari ST per la memorizzazione delle informazioni. Esistono diverse parole chiave nel BASIC ST che vi permettono di salvare le informazioni su disco in modo che possiate recuperarle in un secondo momento, e una volta che avete messo le informazioni nella memoria del computer, potete manipolare numeri e informazioni come volete.

L'istruzione OPEN

Per prima cosa, dovete far posto su disco per un file. Usate l'istruzione OPEN, il cui formato è

OPEN *"tipo file",#numero,"nome del file"[,lunghezza]*

se il *"nome del file"* non esiste, il BASIC ST creerà il file; altrimenti renderà il file disponibile.
Ecco delle brevi descrizioni dei parametri che seguono OPEN.

Parametri di OPEN

Type Sarà costituito dalle lettere O, I oppure R racchiuse tra virgolette. O, che sta per output, significa che manderete le informazioni ad un file sequenziale; I, che sta per input, significa che riceverete informazioni da un file sequenziale; R significa che lavorerete con un file ad accesso diretto, eseguendo l'input o l'output. È importante che battiate la lettera che scegliete in caratteri maiuscoli e che la mettiate tra virgolette.

Number I valori del parametro *number* possono andare da 1 a 15 e devono essere preceduti dal simbolo di numero (#). Per ogni file che aprite dovrete usare un nuovo numero di file; se avete 10 file aperti, quindi, dovranno essere numerati da 1 a 10 in modo che il computer possa trattarli singolarmente. Il numero di file differenzia i diversi file con cui state lavorando.

File Name Parametro che stabilisce il nome del file che volete aprire o specifica il nome del file che volete caricare.

Length Length, parametro opzionale, si misura in byte. Il valore di default è 128, ma per il vostro file potete scegliere la lunghezza che desiderate.

L'istruzione CLOSE

L'opposto di OPEN è CLOSE e il suo formato è

CLOSE *[#]numero[,numero,...]*

dove #numero è il numero del file che volete chiudere. Per esempio, per chiudere il file numero 4 (come specificato dal comando OPEN), battete

```
CLOSE #4
```

Potreste anche battere

```
CLOSE 4
```

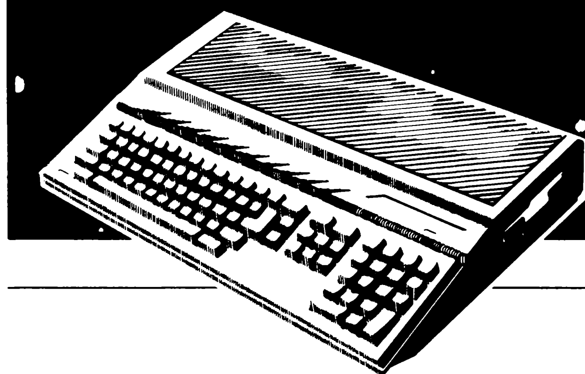
dal momento che il simbolo di numero non è necessario con questa istruzione. CLOSE è necessario per chiudere la linea di comunicazione fra il computer e quel file di disco; doveste includerlo nei vostri programmi quando pensate di non usare più un particolare file.

Le funzioni LOF e LOC

LOF e LOC sono altre due funzioni che potreste trovare utili nella programmazione con i file. LOF vi dice quanto è lungo un file (in byte); una volta che avete aperto un file (per esempio, #3) potreste scoprire quanto è lungo battendo PRINT LOF (#3). LOC vi darà il numero dell'ultimo record di un file con cui avete lavorato. Se state lavorando con i file ad accesso diretto, LOC vi dirà quale è l'ultimo record che avete inserito in un file o che avete prelevato da un file. Per quanto riguarda i file sequenziali, LOC vi dirà quanti record avete letto o scritto in un file dal momento che l'avete aperto.

COME USARE I FILE NEI VOSTRI PROGRAMMI

Avere a disposizione una memoria permanente - i dischi - rende i vostri programmi molto più utili. Applicazioni come tenere un elenco d'indirizzi, conservare dei semplici data base, memorizzare i punteggi più alti dei giochi e salvare i voti degli studenti sono tutte possibili e, con un pò di immaginazione, non sarà difficile pensarne un'altra dozzina. Usate il tipo di file che meglio si addice alle vostre necessità - o i file sequenziali o i file ad accesso diretto - e preoccupatevi di tenere in ordine sia i numeri dei record che i tipi di variabile. Dominare il computer significa aumentare le potenzialità del vostro Atari e adesso che conoscete queste parole chiave per l'input e l'output, sarete in grado, con i vostri programmi BASIC ST, di conquistare nuovi spazi.



PROGRAMMI E PROBLEMI

Per la maggior parte di questo libro, avete imparato le parole chiave del BASIC ST e le tecniche di programmazione. È arrivato il momento di sottoporvi a delle verifiche per vedere quanto avete assimilato. Tuttavia, diversamente dalla maggior parte dei test, questo sarà probabilmente molto divertente e una volta finito avrete cinque programmi da mostrare agli amici come creazioni vostre.

Questo capitolo contiene cinque programmi in BASIC ST già pronti. Prima di listare il programma, descriverò gli obiettivi, abbozzerò l'algoritmo e svilupperò il programma passo per passo. I vari passaggi corrispondono a quelli che concretamente si seguono per sviluppare un programma. Innanzitutto voi, programmatori, dovete avere un'idea di ciò che il programma dovrà fare e di ciò che dovrà fare il computer per raggiungere lo scopo. Successivamente, dovrete elaborare mentalmente o per iscritto uno schema generale del programma in modo che possiate individuare i diversi "blocchi" che dovrete creare per costruire il programma. Infine potrete sedervi a scrivere il programma, migliorandolo e depurandolo dagli errori in modo che possa funzionare perfettamente.

Anche se non intendo proporvi una preparazione troppo laboriosa prima di iniziare a scrivere il programma, un minimo di progettazione è indispensabile per un programma efficiente. Quasi chiunque potrebbe scrivere un programma dopo sole alcune ore di studio sul computer, ma non sono molte le persone che vi dedicano tutto il tempo necessario per scriverne uno buono. Ciò che distingue un buon programma da uno mediocre è l'efficienza e la pulizia (la mancanza di errori di programmazione, i bug). Pensare un pò a come scrivere il programma prima di mettervi alla tastiera vi aiuterà a rendere il risultato finale più efficiente (grazie ad una buona struttura e al supporto della progettazione) e più pulito (se avete le idee chiare su come procedere, è meno facile che nel vostro programma si insinuino degli errori).

Per ciascun programma di questo capitolo, il paragrafo intitolato Descrizione del programma lo descriverà abbastanza dettagliatamente da permettervi di scriverlo da voi senza consultare il listato. Siete liberi di consultare i capitoli precedenti del libro e il *Manuale del BASIC ST*, ma dovreste farvi un punto d'onore di completare il programma basandovi solo sulla descrizione senza sbirciare la mia versione.

Se vi pare di aver proprio bisogno di ulteriori suggerimenti, consultate il paragrafo intitolato Componenti. Vi troverete uno schema generale di come deve essere strutturato il programma ed alcune raccomandazioni generali circa le parole chiave e le tecniche a cui dovete ricorrere per svilupparlo. Se volete potete controllare solo una o due parti di questa sezione, oppure potete leggerla interamente.

Se tutto questo ancora non basta, potete andare alla sezione chiamata Sviluppo del programma, in cui vi guido passo per passo nella creazione. Le linee non saranno necessariamente in ordine poiché svilupperò le subroutine

specifiche man mano (e non secondo l'ordine finale). Vi accorgerete che quando sviluppate i vostri programmi, dovrete lavorare per moduli e stadi non consecutivi e mentre li state creando dovrete apportarvi continue modifiche. Quando avrete finito questa sezione, dovreste avere un programma che funziona.

La sezione dedicata allo sviluppo del programma è un listato completo del programma; questo renderà più semplice controllare un qualsiasi errore e inoltre vi servirà come termine di paragone se ne state scrivendo uno vostro. Dopo ogni listato ho inserito anche la sezione Miglioramenti Proposti per darvi un'idea di come migliorare un programma; potete provare alcuni di questi suggerimenti o crearne voi per imparare a programmare sempre meglio con il BASIC ST. In questo campo, l'esperienza è sicuramente l'insegnante migliore.

Scrivendo il vostro programma, non pensate che sia sbagliato solo perché è scritto in un modo diverso dal mio. Se il vostro rientra nei limiti della descrizione del programma (o comunque si avvicina ad essa) e se gira senza problemi, potete congratularvi con voi stessi per avere risolto brillantemente il problema. Potrebbe anche succedere che il vostro programma sia strutturato in modo più ingegnoso e sia più divertente da usare che non il mio, oppure che sia scritto in modo tanto diverso che sarebbe inutile ogni confronto con il mio. In ogni caso, lo scopo di questo capitolo è di permettervi di usare - e verificare - quanto avete imparato seguendo certe direttive per la programmazione. Meno suggerimenti vi servono, più sarete in grado di sviluppare da soli l'idea base in un programma. Cercate di usare il meno possibile i miei suggerimenti della sezione Componenti; il programma sarà veramente vostro se lo completate in base alle vostre idee.

IL GIOCO PER RAGAZZI ALTO/BASSO

Il primo programma è un semplice gioco per ragazzi, facile da scrivere usando il BASIC ST e un po' di logica.

Descrizione del programma

Scrivete un programma che selezioni un numero a caso da 1 a 100 che l'utente dovrà indovinare. Dopo ogni tentativo d'indovinare il numero, il computer dirà se è troppo alto, troppo basso o se è quello giusto. Se la risposta è alto o basso, il computer visualizza nell'angolo in alto a destra dello schermo *HI* o *LO* rispettivamente. Dovreste rendere il programma amichevole, per cui quando il giocatore indovina il numero, fate in modo che il computer vi-

sualizzi le sue congratulazioni ed un punteggio basato sul numero dei tentativi fatti per arrivare al numero esatto.

Componenti

Le cinque parti del programma sono le seguenti:

1. All'inizio del programma, scrivete una routine che pulisca lo schermo, saluti il giocatore e gli chieda se gli piacerebbe giocare.
2. Fate in modo che il BASIC ST selezioni un numero a caso fra 1 e 100 con la funzione RND e usi INPUT per ricevere le risposte del giocatore.
3. Il programma può dire al giocatore per ciascun tentativo se è troppo alto o troppo basso (usando il simbolo > per maggiore di e < per minore di). Usate le istruzioni LINEF per stampare HI o LO nell'angolo in alto a destra dello schermo.
4. Se il giocatore indovina, il computer stampa sullo schermo una moltitudine di *ESATTO!!!*; se il tentativo è alto o basso, il computer accetterà ulteriori tentativi del giocatore con INPUT.
5. Alla fine del programma il computer userà i test logici > e < per valutare la prestazione del giocatore. Alla fine il programma chiede se il giocatore ha intenzione di giocare nuovamente e a seconda della risposta prende i provvedimenti del caso.

Sviluppo del programma

Per prima cosa il computer ha bisogno di una routine che indovini un numero a caso fra 1 e 100. Come avrete già immaginato, il generatore di un numero qualunque darà lo stesso numero ogni volta che il computer viene acceso o che viene fatto girare un nuovo programma. Per cui, se batteste *PRINT RND*, scriveste il risultato, spegneste il computer e poi lo riaccendeste e batteste nuovamente *PRINT RND*, otterreste esattamente lo stesso numero. Per fare in modo che questo numero "a caso" sia ancora più casuale, il computer prima chiederà al giocatore d'inserire un numero qualunque fra 1 e 500, che viene memorizzato nella variabile RN. Se il numero inserito è minore di 1 o maggiore di 500, il programma chiederà all'utente di battere un numero che rientri entro i limiti specificati.

```
50 PRINT: PRINT "Sto pensando ad un numero fra 1 e 100, "
```

```
60 PRINT "ma prima ho bisogno di avere un numero da te. Batti un
   numero "
70 PRINT "fra 1 e 500 e premi Return."
80 INPUT "Qual è il numero ",RN: IF RN<1 OR RN>500 THEN
   PRINT "Per piacere sceglilo fra 1 e 500": GOTO 80
```

Una volta che il computer ha ricevuto il numero del giocatore, il programma lo usa in un'istruzione RANDOMIZE, che reimposta il generatore di numeri casuali in modo che i numeri siano meno prevedibili. La variabile CN (numero scelto) memorizza il risultato di 100*RND, che è un numero a caso fra 1 e 100.

```
90 PRINT "Grazie!": PRINT: RANDOMIZE RN: CN=INT (100*RND)
```

Prima di cominciare il gioco, il programma si ferma per un momento e usa il loop FOR ... NEXT, cancella lo schermo, quindi chiede al giocatore di inserire un numero. Questo tentativo viene memorizzato nella variabile PG (tentativo del giocatore), che tra breve verrà confrontato a CN.

```
100 FOR D=1 TO 500: NEXT
110 CLEARW 2
120 INPUT "Quale pensi che sia il mio numero ";PG
```

Se il numero del giocatore è minore di 1 o maggiore di 100, il computer dirà al giocatore di mantenersi entro i limiti 1-100; se il tentativo è all'interno dei limiti, la variabile che conta il numero di tentativi (NG) viene aumentata di 1. Notate che se il giocatore accidentalmente fa un tentativo uscendo dai limiti, il computer ritorna alla linea 120 senza aumentare la variabile NG dopo essere andato alla linea 140.

```
130 IF PG<1 OR PG>100 THEN PRINT "Mantieni i tuoi numeri fra 1
    e 100. Non contero' questo tentativo a tuo sfavore.":GOTO 120
140 NG=NG+1
```

Adesso il programma può confrontare i tentativi del giocatore con il numero del computer. Se PG è maggiore di CN, il programma va alla subroutine della linea 180 che disegna un grande *HI* nell'angolo in alto a destra dello schermo. La variabile FL ha una particolare importanza, dal momento che definisce il colore di *HI.* Ciò è importante perché *HI* deve apparire solo per alcuni istanti e poi viene cancellato. Non c'è una parola chiave nel BASIC ST che possa cancellare solo una certa parte dello schermo, per cui l'unica soluzione è quella di disegnare nuovamente *HI* nel colore di sfondo; l'effetto sarà lo stesso

della cancellatura: *HI* diventa invisibile. Alla linea 180, FL è impostata a 2, il che significa che il primo *HI* sarà rosso.

```
150 IF PG>CN THEN GOTO 180
180 FL=2: REM routine alto
190 COLOR 1,1,FL
```

Le prossime quattro istruzioni LINEF tracciano le quattro linee che servono per creare *HI* (tre per la H e uno per la I). Non è stato difficile risalire a queste coordinate, poiché ho dovuto semplicemente visualizzare le relazioni dei punti che avrebbero formato queste due lettere.

```
200 LINEF 400,10,400,50
210 LINEF 400,30,430,30
220 LINEF 430,10,430,50
230 LINEF 450,10,450,50
```

Il loop NEXT ... FOR alla linea 240 dà alla variabile FL un altro obiettivo cioè che il computer conti da 1 a FL*1000. Quando FL è uguale a 2 (il che significa che *HI* è visibile), il computer si ferma per un attimo mentre sta contando da 1 a 2000; questo farà in modo che *HI* rimanga sullo schermo il tempo necessario perché il giocatore riesca a leggerlo. Dopo che questo loop è stato completato per la prima volta, il valore di FL viene trasformato in 0, e il computer ritorna alla linea 190 in modo che possa nuovamente tracciare *HI*, questa volta usando il colore dello sfondo, il bianco. La seconda volta che si incontra la linea 240, FL è uguale a 0, il che significa che il loop FOR ... NEXT non farà fermare il programma e, dal momento che FL non è uguale a 2, il computer andrà alla linea 250, che gli dirà di tornare alla linea 120 (per ricevere il numero successivo dal giocatore).

```
240 FOR D=1 TO FL*1000: NEXT: IF FL=2 THEN FL=0: GOTO 190
250 GOTO 120
```

La routine per un numero troppo basso è quasi identica. Le uniche differenze sono che il colore viene prima impostato sul rosso (solo per dare al programma un po' di varietà) e che le istruzioni LINEF hanno delle coordinate diverse, dal momento che il programma traccia la parola *LO* invece di *HI*. Infine, poiché FL è in un primo tempo uguale a 3 anziché a 2, il loop FOR .. NEXT usato per dare ua pausa al programma conta da 1 a FL*750 invece che da 1 a FL*1000; in questo modo il computer conterà fino a 2250, una pausa più che sufficiente.

```
160 IF PG<CN THEN GOTO 260
260 FL=3: REM routine basso
270 COLOR 1,1,FL
280 LINEF 400,10,400,50
290 LINEF 400,50,430,50
300 LINEF 450,10,450,50
310 LINEF 450,50,500,50
320 LINEF 500,50,500,10
330 LINEF 450,10,500,10
340 FOR D=1 TO FL*750: NEXT: IF FL=3 THEN FL=0: GOTO 270
350 GOTO 120
```

Quando il giocatore indovina il numero del computer, il programma va alla linea 360 dove stampa 48 volte *ESATTO!!!* sullo schermo. Poi il programma pulisce di nuovo lo schermo e dice al giocatore quanti tentativi ha fatto prima di indovinare il numero esatto.

```
170 IF PG=CN THEN GOTO 360
360 CLEARW 2: FOR D=1 TO 48: PRINT "ESATTO!!!";: NEXT
370 CLEARW 2: PRINT "Hai indovinato dopo ";NG;" tentativi."
```

I parametri di valutazione sono piuttosto arbitrari e probabilmente troppo difficili per un bambino, per cui potreste riadattarli dopo avere provato il programma alcune volte. Se indovina dopo meno di 3 tentativi, il programma stampa *Sei bravissimo!* Se indovina dopo più di 2 ma meno di 6 tentativi la scritta sarà *Non male* e con più di 6 tentativi stamperà *Continua ad esercitarti.* Notate che alla linea 390 sono combinate due espressioni logiche per scoprire se il numero dei tentativi cade nell'intervallo di altri due numeri.

```
380 IF NG<3 THEN PRINT "Sei bravissimo!"
390 IF NG>2 AND NG<6 THEN PRINT "Non male."
400 IF NG=>6 THEN PRINT "Continua ad esercitarti..."
```

Il programma adesso chiede se il giocatore vuole giocare ancora. Se il primo carattere della risposta è *Y* o *y*, il programma va alla linea 50 per ricominciare il gioco. Altrimenti, lo schermo viene cancellato, il computer dice *Ciao!* e il programma è finito.

```
410 PRINT: INPUT "Vuoi giocare ancora";AN$
420 IF LEFT$(AN$,1)="S" OR LEFT$(AN$,1)="s" THEN GOTO 50
    ELSE CLEARW 2: PRINT "Ciao!": END
```

Il tocco finale da aggiungere è un saluto che mette a suo agio il giocatore

quando prova per la prima volta il programma. Alla linea 30, il programma chiede se il giocatore vuole fare un gioco; se il primo carattere della risposta è *S* o *s*, l'ST stampa *Benissimo!* e va alla linea 50. Altrimenti, il computer, deluso, risponde *Allora, sarà per un'altra volta* e chiude il programma.

```
10 REM Alto/Basso Gioco per ragazzi
20 FULLW 2: CLEARW 2: GOTOXY 0,0
30 INPUT "Ciao! Vuoi giocare con me ";AN$
40 IF LEFT$(AN$,1)="S" OR LEFT(AN$,1)="s" THEN PRINT
   "Benissimo!" ELSE PRINT "Allora, sarà per un'altra volta...": END
```

La figura 8.1 mostra un campione di questo programma e il listato 8.1 presenta l'intero programma High/Low.

Miglioramenti proposti

- Fate in modo che possa variare il livello del gioco in modo che gli intervalli da cui estrarre il numero casuale e i parametri di valutazione possano essere adattati al grado di abilità del giocatore.
- Cercate di rendere più vivaci gli effetti di *AL, BA* ed *ESATTO!!!*. Potete provare a cambiare i colori, a tracciare dei grafici più elaborati ed eventual-

```
RUN
Ciao! Vuoi giocare con me? No
Allora, sara' per un'altra volta...
RUN
Ciao! Vuoi giocare con me? Si
Benissimo! Sto pensando ad un numero fra 1 e 100,
ma prima ho bisogno di avere un numero da te. Batti un numero
fra 1 e 500 e premi Return.
Qual e' il numero? 653
Per piacere sceglilo fra 1 e 500
Qual e' il numero? 420
Grazie!
Quale pensi che sia il mio numero? 105
Mantieni i tuoi numeri fra 1 e 100. Non contero' questo tentativo
a tuo sfavore.
Quale pensi che sia il mio numero? 50
Quale pensi che sia il mio numero? 25
Quale pensi che sia il mio numero? 35
Quale pensi che sia il mio numero? 33
ESATTO!!!ESATTO!!!ESATTO!!!ESATTO!!!ESATTO!!!ESATTO!!! (ecc.)
Hai indovinato dopo 4 tentativi.
Non male.
Vuoi giocare ancora? Non ora
Ciao!
```

Figura 8.1: un esempio di esecuzione del programma Gioco alto/basso.

```
10  REM Alto/basso Gioco per ragazzi
20  FULLW 2: CLEARW 2: GOTOXY 0,0
30  INPUT "Ciao! Vuoi giocare con me";AN$
40  IF LEFT$(AN$,1)="S" OR LEFT(AN$,1)="s" THEN PRINT "Benissimo!"
    ELSE PRINT "Allora, sara' per un'altra volta...": END
50  PRINT: PRINT "Sto pensando ad un numero fra 1 e 100,"
60  PRINT "ma prima ho bisogno di avere un numero da te. Batti un
    numero"
70  PRINT "fra 1 e 500 e premi Return."
80  INPUT "Qual e' il numero",RN: IF RN<1 OR RN>500 THEN
    PRINT "Per piacere sceglilo fra 1 e 500": GOTO 80
90  PRINT "Grazie!": PRINT: RANDOMIZE RN: CN=INT (100*RND)
100  FOR D=1 TO 500: NEXT
110  CLEARW 2
120  INPUT "Quale pensi che sia il mio numero ";PG
130  IF PG<1 OR PG>100 THEN PRINT "Mantieni i tuoi numeri fra 1 e
     100. Non contero' questo tentativo a tuo sfavore.":GOTO 120
140  NG=NG+1
150  IF PG>CN THEN GOTO 180
160  IF PG<CN THEN GOTO 260
170  IF PG=CN THEN GOTO 360
180  FL=2: REM (routine alto)
190  COLOR 1,1,FL
200  LINEF 400,10,400,50
210  LINEF 400,30,430,30
220  LINEF 430,10,430,50
230  LINEF 450,10,450,50
240  FOR D=1 TO FL*1000: NEXT: IF FL=2 THEN FL=0: GOTO 190
250  GOTO 120
260  FL=3: REM (routine basso)
270  COLOR 1,1,FL
280  LINEF 400,10,400,50
290  LINEF 400,50,430,50
300  LINEF 450,10,450,50
310  LINEF 450,50,500,50
320  LINEF 500,50,500,10
330  LINEF 450,10,500,10
340  FOR D=1 TO FL*750: NEXT: IF FL=3 THEN FL=0: GOTO 270
350  GOTO 120
360  CLEARW 2: FOR D=1 TO 48: PRINT "ESATTO!!!";: NEXT
370  CLEARW 2: PRINT "Hai indovinato dopo ";NG;" tentativi."
380  IF NG<3 THEN PRINT "Sei bravissimo!"
390  IF NG>2 AND NG<6 THEN PRINT "Non male."
400  IF NG=>6 THEN PRINT "Continua ad esercitarti..."
410  PRINT: INPUT "Vuoi giocare ancora";AN$
420  IF LEFT$(AN$,1)="S" OR LEFT$(AN$,1)="s" THEN GOTO 50 ELSE
     CLEARW 2: PRINT "Ciao!": END
```

Listato 8.1: programma Gioco alto/basso

mente ad aggiungere degli effetti sonori.

- Personalizzate il programma in modo che, dopo che il giocatore ha battuto il suo nome, il programma usi il suo nome in tutti i prompt e negli altri messaggi che visualizza (per esempio, *Quale pensi che sia il mio numero, Andrea?).*
  Questo renderà il gioco molto più divertente.

IL PROGRAMMA PER DIAGRAMMA CIRCOLARE

Nel Capitolo 4 avete visto come creare un diagramma circolare usando dei valori casuali. Questo programma sarà molto più utile, poiché traccia un diagramma in base ai dati numerici reali che voi inserirete.

Descrizione del programma

Scrivete un programma che accetti dall'utente fino a 15 numeri e stringhe di testo. Una volta che il programma ha questi dati, li visualizzerà nella forma di un diagramma circolare assegnando ad ogni settore le dimensioni appropriate. L'intero cerchio verrà riempito e ogni settore del grafico sarà di colore diverso che sarà lo stesso usato per stampare la rispettiva etichetta di testo. Per esempio, se voleste rappresentare gli uomini e le donne di un paese e i dati fossero 48 per cento e 52 per cento, il 52 per cento del cerchio potrebbe essere verde e sullo schermo la parola *Donne* apparirà in verde, mentre il rimanente 48 per cento del cerchio potrebbe essere rosso con la parola *Uomini* scritta in rosso sullo schermo. Questa chiave rende il diagramma circolare più utile, dal momento che le etichette possono essere facilmente associate ai settori circolari.

Componenti

1. Usate una matrice per memorizzare i dati.
2. Ogni dato dovrebbe avere una corrispondente etichetta di testo.
3. Usate la routine GDP del VDISYS che traccia i settori pieni del cerchio.
4. Tutti i dati devono essere sommati e poi divisi per 3600 per sapere quanto devono essere ampi gli angoli per ogni settore. L'intero cerchio comprende 3600 decimi di grado (l'unità con cui la routine VDISYS misura gli angoli), e dividendo 3600 per il totale di tutti i dati, siete in grado di determinare il moltiplicatore per calcolare la misura dell'angolo di ciascun settore. Per esempio, supponete di avere tre dati: 40, 50 e 70, in totale 160. 3600 diviso 160 dà 22,5, il che significa che ogni dato deve essere moltiplicato per 22,5 per trovare quante unità il suo settore dovrebbe occupare (in questo caso, i tre dati richiedono 900, 1125 e 1575 decimi di grado, la cui somma è 3600 decimi di grado: il cerchio completo).

Sviluppo del programma

Per prima cosa il programma ha bisogno di ricevere dei dati dalla tastiera. Questi dati verranno memorizzati nella matrice D(), che sarà di dimensioni atte a ricevere fino a 15 diversi elementi.

```
10 REM programma per diagramma circolare
20 FULLW 2: CLEARW 2: COLOR 1,1,1
30 DIM D(15)
```

La variabile PD conterà il numero delle voci che sono state inserite. Finché PD è minore di 15 (il numero massimo di dati), il computer aumenterà di 1 il valore attuale di PD. Una volta che è stato inserito il quindicesimo dato, il programma salterà alla linea 70. Qui il computer userà un loop NEXT ... FOR per trovare il valore totale di tutti i dati che si trovano nella sua memoria.

```
40 IF PD<15 THEN PD=PD+1 ELSE GOTO 70
70 FOR I=1 TO PD: TL=TL+D(I):NEXT
```

Il computer accetta i dati con l'istruzione INPUT; prima viene inserito il dato seguito da una virgola e poi la stringa che corrisponde al dato. Per esempio, dopo che il programma ha chiesto il dato numerico e la sua stringa, potreste inserire *400,Vendite Gennaio*:

```
50 PRINT "Inserisci il dato numero ";PD;" e la sua etichetta": INPUT
   D(PD),IT$(PD)
```

Se il numero del dato inserito come D(PD) è uguale a 0 (il che indica che volete tracciare il diagramma circolare) o se PD raggiunge 15, il programma procederà alla linea 70 per tracciare il grafico; inoltre, il numero dei dati viene diminuito di 1 (dal momento che l'ultimo dato inserito non è rilevante). Se nessuna di queste due condizioni viene soddisfatta, il computer supporrà che vogliate inserire un altro dato, quindi andrà alla linea 40 per ricevere un altro input.

```
60 IF D(PD)=0 OR PD=15 THEN PD=PD—1 ELSE GOTO 40
```

Le linee del programma che tracciano il diagramma circolare sono costruite attorno alla routine GDP del VDISYS che traccia i settori circolari pieni. Per prima cosa, le istruzioni POKE dovranno impostare la routine GDP del VDISYS:

```
 80 CLEARW 2: POKE CONTRL,11
 90 POKE CONTRL+2,4
100 POKE CONTRL+6,2
110 POKE CONTRL+10,3
```

Il centro del cerchio è fissato a (220,100), e il raggio (definito alla posizione di byte PTSIN+12) è di 70 pixel.

```
120 POKE PTSIN,220: POKE PTSIN+2,100
130 FOR I=2 TO 7: POKE PTSIN+I*2,0: NEXT
140 POKE PTSIN+12,70
```

La variabile AI (incremento dell'angolo) misura i settori circolari in modo che insieme formino un cerchio completo. Dal momento che gli angoli nella routine GDP del VDISYS vengono misurati in decimi di grado, un cerchio ha 3600 decimi. Dividendo 3600 per il valore totale dei dati, il programma ricava un numero fisso che moltiplicato per il valore dei singoli dati consente di determinare il valore in decimi di grado di ogni settore.

```
150 AI=3600/TL
```

Il programma adesso deve esaminare i dati dal primo all'ultimo (PD). La variabile SA verrà usata come angolo iniziale, che si trova dove il computer comincia a tracciare il settore attuale. Quando comincia questa parte del programma SA sarà uguale a 0. Quando i settori saranno stati tracciati, invece, SA verrà aumentata in modo che ogni angolo di ogni nuovo settore cominci da dove è finito l'ultimo settore. Comunque, ogni settore andrà da SA (sempre in decimi di grado) a SA+D(I)*AI; questa formula, per l'angolo finale, aggiunge semplicemente l'angolo iniziale al dato moltiplicato per AI, il che stabilisce esattamente la misura di ogni settore.

```
155 SA = 0
160 FOR I=1 TO PD
170 POKE INTIN,SA
180 POKE INTIN+2,SA+D(I)*AI
```

Adesso inserite le istruzioni POKE per gli angoli iniziale e finale del settore per la routine GDP del VDISYS. L'angolo iniziale (SA) adesso deve essere aumentato in modo che il prossimo settore tracciato possa cominciare dall'angolo corretto. SA deve essere uguale a se stesso più D(I)*AI (l'angolo finale per il settore che si sta tracciando al momento), quindi il computer controlla per vedere se il valore I nel loop NEXT ... FOR è uguale PD. Se è così, vuol dire che il computer ha raggiunto il suo ultimo dato per cui il programma

usa un'istruzione POKE per inserire 3600, l'angolo finale per completare il cerchio, in INTIN+2:

```
190 SA=SA+D(I)*AI: IF I=PD THEN POKE INTIN+2,3600
```

Prima di tracciare il settore, il computer stabilisce il colore sia del testo che del fondo con il numero della variabile contatore I in FOR...NEXT. VDISYS (1) alla linea 200 traccia il settore completo sullo schermo e il programma usa la linea 210 per visualizzare il testo relativo a quel dato. Se stessimo trattando le vendite di varie periferiche per computer e le stampanti fossero il 22 per cento delle vendite, la stringa "Stampanti=22%" apparirà nello stesso colore del settore corrispondente. Inoltre, queste stringhe di testo vengono posizionate usando GOTOXY e il valore di I in modo che appaiano ben ordinate. L'ultima coppia di linee del programma chiude il loop e crea un loop infinito alla linea 230 per conservare il contenuto dello schermo in modo che voi possiate esaminarlo.

```
200 COLOR I,I: VDISYS (1)
210 GOTOXY 0,I: PRINT IT$(I);"—";INT(100*D(I)/TL)"%"
220 NEXT
230 GOTO 230
```

La figura 8.2 mostra un diagramma circolare prodotto dai dati che seguono.

```
RUN
Inserisci il dato numero 1 e la sua etichetta
? 57,Stampanti
Inserisci il dato numero 2 e la sua etichetta
? 103,Modem
Inserisci il dato numero 3 e la sua etichetta
? 43,Monitor monocromatici
Inserisci il dato numero 4 e la sua etichetta
? 79,Monitor a colori
Inserisci il dato numero 5 e la sua etichetta
? 39, Libri
Inserisci il dato numero 6 e la sua etichetta
? 0,X
```

Miglioramenti proposti

- Aggiungete un'opzione che vi permetta di cambiare il centro e il raggio del diagramma circolare.

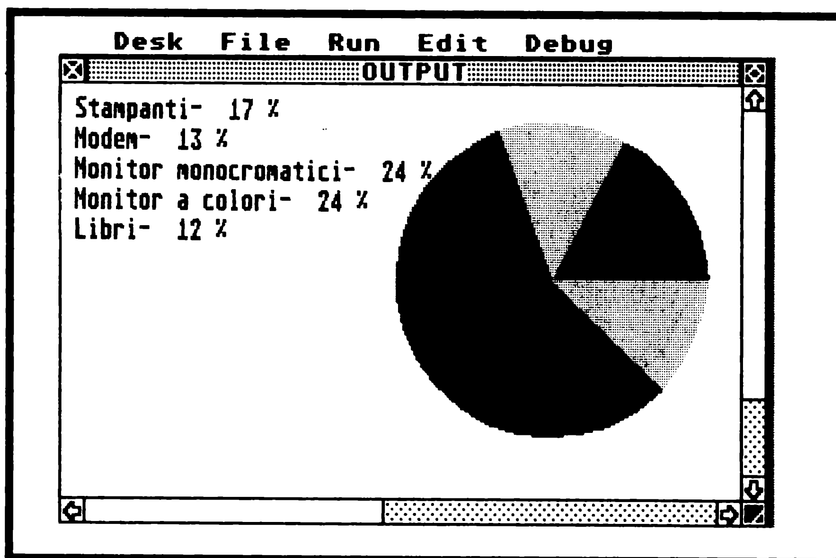


Figura 8.2: un ipotetico diagramma circolare che visualizza le vendite di periferiche per computer.

- Usate dei colori di fondo diversi con l'istruzione COLOR mentre tracciate i settori.

IL PROGRAMMATORE DEI COLORI

Trovare un particolare colore fra i 512 disponibili può essere un'impresa difficile. Il Programmatore dei Colori semplifica questo compito mettendovi in grado di usare il tastierino numerico per scorrere la vasta gamma di colori e trovare quello che vi interessa.

Descrizione del programma

Un cerchio pieno nel mezzo dello schermo mostra il colore che l'utente intende cambiare. Per alterare il colore usate la routine 14 del VDISYS (descritta al Capitolo 6) e usate il tastierino numerico come dispositivo di input per cambiare i livelli del rosso, del verde e del blu. I tasti 7,8 e 9 aumenteranno il livello del rosso, del verde e del blu; i tasti 4, 5 e 6 riporteranno i rispettivi livelli al loro valore medio di 500; e i tasti 1, 2 e 3 diminuiranno i rispettivi livelli. Poiché ciascuno dei livelli può variare da 1 a 1000, quando il program-

ma inizia dovreste impostarli a 500. Questo programma permette all'utente di aumentare o diminuire il livello di 10 unità per volta e permette di avere il valore dei tre livelli costantemente visualizzati nella parte superiore dello schermo. Per uscire dal programma, bisogna premere la Q, maiuscola o minuscola.

Componenti

1. Usate le variabili R, G e B per memorizzare gli attuali livelli dei colori.
2. La funzione INP (2) riceverà informazioni dalla tastiera in modo che il tastierino numerico possa essere usato come dispositivo d'input e che anche il tasto Q possa essere controllato.
3. Usate il comando ON ... GOSUB in modo che il computer possa andare alla subroutine appropriata a seconda del tasto premuto sul tastierino numerico.
4. Aggiungete tre linee per essere sicuri che le intensità del rosso, del verde e del blu non vadano al di sotto di 1 o al di sopra di 1000.
5. Usate PCIRCLE per tracciare il cerchio pieno che assumerà il colore che l'utente vuole cambiare.

```
10 REM Programma del diagramma circolare
20 FULLW 2: CLEARW 2: COLOR 1,1,1
30 DIM D(15)
40 IF PD<15 THEN PD=PD+1 ELSE GOTO 70
50 PRINT "Inserisci il dato numero ";PD;" e la sua etichetta": INPUT
D(PD),IT$(PD)
60 IF D(PD)=0 THEN PD=PD-1: GOTO 40
70 FOR I=1 TO PD: TL=TL+D(I): NEXT
80 CLEARW 2: POKE CONTRL,11
90 POKE CONTRL+2,4
100 POKE CONTRL+6,2
110 POKE CONTRL+10,3
120 POKE PTSIN,220: POKE PTSIN+2,100
130 FOR I=2 TO 7: POKE PTSIN+I*2,0: NEXT
140 POKE PTSIN+12,70
150 AI=3600/TL
155 SA=0
160 FOR I=1 TO PD
170 POKE INTIN,SA
180 POKE INTIN+2,SA+D(I)*AI
190 SA=SA+D(I)*AI: IF I=PD THEN POKE INTIN+2,3600
200 COLOR I,I: VDISYS (1)
210 GOTOXY 0,I: PRINT IT$(I);"-  ";INT(100*D(I)/TL)"%"
220 NEXT
230 GOTO 230
```

Listato 8.2: programma del diagramma circolare

Sviluppo del programma

Per prima cosa avete bisogno di scrivere una routine che accetti l'input dalla tastiera e che cambi le variabili R, G e B. I valori ASCII per i tasti del tastierino numerico vi vengono dati nella tabella 8.1.

X = INP (2) memorizzerà i valori ASCII ricevuti dalla tastiera; se X è uguale a 81 o a 113 (che sono Q o q, e dicono al computer di lasciare il programma), il Programmatore dei colori finisce. Altrimenti, il programma usa ON X—48 GOSUB in modo che venga usata la subroutine appropriata a seconda del tasto appena premuto. Queste subroutine cambiano il valore di R, G o B e poi tornano alla linea che segue GOSUB.

```
 70 X=INP (2)
 80 IF X=81 OR X=113 THEN END
 90 ON X—48 GOSUB 160,170,180,190,200,210,220,230,240
160 R=R—10: RETURN
170 G=G—10: RETURN
180 B=B—10: RETURN
190 R=500: RETURN
200 G=500: RETURN
210 B=500: RETURN
220 R=R+10: RETURN
```

Numeri sul tastierino	Codice ASCII	Cambiamento di colore
1	49	Il livello di rosso si abbassa di 10
2	50	Il livello di verde si abbassa di 10
3	51	Il livello di blu si abbassa di 10
4	52	Il livello di rosso torna a 500
5	53	Il livello di verde torna a 500
6	54	Il livello di blu torna a 500
7	55	Il livello di rosso aumenta di 10
8	56	Il livello di verde aumenta di 10
9	57	Il livello di blu aumenta di 10

Tabella 8.1: valori ASCII, cambiamenti di colore e corrispondenti numeri sul tastierino.

```
230 G=G+10: RETURN
240 B=B+10: RETURN
```

Quando il programma esegue il RETURN, esso comincia dalla linea 100 in modo da controllare che le variabili R, G e B all'interno dei loro intervalli. Se una qualsiasi di queste variabili è minore di 0, viene fissata uguale a 0 e, se va oltre 1000, viene fissata uguale a 1000. Una volta che il computer abbia controllato queste tre variabili numeriche, le stampa nella parte superiore dello schermo.

```
100 IF R<0 THEN R=0 ELSE IF R>1000 THEN R=1000
110 IF G<0 THEN G=O ELSE IF G>1000 THEN G=1000
120 IF B<0 THEN B=0 ELSE IF B>1000 THEN B=1000
130 GOTOXY 0,0: PRINT TAB (5);R;TAB (20);G;TAB (35);B
```

All'inizio del programma, i livelli di queste tre variabili dovrebbero essere fissati a 500. Inoltre, vengono date le istruzioni POKE per chiamare la routine VDISYS che cambia il colore e viene tracciato un cerchio pieno usando il colore in modo che l'utente possa vedere il cerchio che cambia di colore.

```
10 REM Programmatore dei colori
20 FULLW 2: CLEARW 2
30 R=500: G=500: B=500
40 POKE CONTRL,14: POKE CONTRL+2,0
50 POKE CONTRL+6,4: POKE INTIN,2
60 COLOR 1,2: PCIRCLE 151,83,70
70 X=INP (2)
80 IF X=81 OR X=113 THEN END
90 ON X-48 GOSUB 160,170,180,190,200.210,220,230,240
100 IF R<0 THEN R=0 ELSE IF R>1000 THEN R=1000
110 IF G<0 THEN G=0 ELSE IF G>1000 THEN G=1000
120 IF B<0 THEN B=0 ELSE IF B>1000 THEN B=1000
130 GOTOXY 0,0: PRINT TAB (5);R;TAB (20);G;TAB (35);B;
140 POKE INTIN+2,R: POKE INTIN+4,G: POKE INTIN+6,B
150 VDISYS (1): GOTO 70
160 R=R-10: RETURN
170 G=G-10: RETURN
180 B=B-10: RETURN
190 R=500: RETURN
200 G=500: RETURN
210 B=500: RETURN
220 R=R+10: RETURN
230 G=G+10: RETURN
240 B=B+10: RETURN
```

Listato 8.3: programma dei colori

```
10 REM Programmatore dei colori
20 FULLW 2: CLERAW 2
30 R=500: G=500: B=500
40 POKE CONTRL,14: POKE CONTRL+2,0
50 POKE CONTRL+6,4: POKE INTIN,2
60 COLOR 1,2: PCIRCLE 151,83,70
```

L'ultima parte del programma cambia il colore. Per usare POKE, mettete le variabili R, G e B nella cella appropriata. L'istruzione VDISYS (1) altera il colore. Il GOTO 70 alla linea 150 manda il programma di nuovo all'istruzione INP in modo che il computer possa continuare a ricevere i comandi dal tastierino numerico.

```
140 POKE INTIN+2,R: POKE INTIN+4,G: POKE INTIN+6,B
150 VDISYS (1): GOTO 70
```

Quando usate il Programmatore dei colori, ricordate che i tre tasti a sinistra sul tastierino numerico cambiano il rosso, i tre tasti in mezzo cambiano il verde e i tre tasti a destra cambiano il blu. Come probabilmente avrete già dedotto scrivendo questo programma, i tasti in alto del tastierino numerico alzano il livello di 10, i tasti in mezzo fanno ritornare il livello a 500 e i tasti in basso diminuiscono l'intensità di 10. Il listato 8.3 mostra il programma completo.

Miglioramenti proposti

- Aggiungete un'opzione che permetta di fissare gli incrementi con cui cambia il livello dei colori (invece di mantenere questi cambiamenti fissi a 10).
- Aggiungete una funzione che aumenti la velocità con cui l'utente può scorrere i diversi livelli del colore.
- Fate scegliere all'utente il colore da cambiare, invece di cambiare sempre il colore numero 2; POKE INTIN determina quale registro di colore verrà cambiato.

IL PROGRAMMATORE DEI SUONI

Il Programmatore dei suoni funziona quasi allo stesso modo del programma per i colori, poiché usa il tastierino numerico come dispositivo di input. Usando il tastierino numerico, potete cercare fra tanti suoni quello giusto.

Descrizione del programma

Questo programma dovrebbe permettere all'utente di cambiare la forma, l'ottava, la nota e il periodo direttamente dal tastierino numerico, e anche di passare dal rumore al suono. Q indica che l'utente vuole uscire dal programma e le istruzioni WAVE e SOUND dovrebbero essere usate per produrre e alterare il suono. I valori per gli argomenti ottava, nota, forma, periodo e attivatore (1 per un suono, 8 per un rumore) di queste istruzioni dovrebbero comparire nella parte alta dello schermo, in modo che il programma sia utile ad una persona che stia cercando un certo rumore o una certa nota.

Componenti

1. All'inizio del programma, fissate il periodo a 1000, la forma ad 8, e l'attivatore a 8 nell'istruzione WAVE; fissate la nota a 3 e l'ottava a 3 nell'istruzione SOUND.
2. Usate WAVE con gli argomenti attivatore, forma e periodo e SOUND con gli argomenti nota ed ottava. Inoltre, quando l'utente vuole ottenere un rumore anziché una nota, non c'è bisogno di usare l'istruzione SOUND.
3. Non risparmiate sulle routine di controllo degli errori, in modo da essere sicuri che nessuno dei valori ecceda i suoi limiti.
4. La funzione INP va impiegata per ottenere il valore ASCII del tasto del tastierino che avete premuto. Anche ON...GOSUB è utile per accedere all'appropriata routine per cambiare qualsiasi valore l'utente voglia alterare.

Sviluppo del programma

Il cuore di questo programma è la routine d'input dal tastierino. Prima di tutto il computer ottiene il valore ASCII dalla tastiera con l'istruzione INP, poi controlla se quel valore è uguale a 81 o a 113; in caso affermativo, significa che è stato premuto il tasto Q ed il computer cancella lo schermo, disattiva il canale acustico e chiude il programma.

```
50 X=INP (2)
60 IF X=81 OR X=113 THEN CLEARW 2: SOUND 1,0: WAVE 0:
   END
```

Se X è un valore diverso da quelli corrispondenti al tasto Q, dirigerà il programma verso la subroutine che l'utente ha richiesto. La tabella 8.2 elenca le

Numeri sul tastierino	Codice ASCII	Funzione
1	49	Ottava aumenta di 1
2	50	Ottava diminuisce di 1
3	51	Passa da rumore a suono
4	52	Busta diminuisce di 1
5	53	Periodo diminuisce di 1
6	54	Nota si abbassa di 1
7	55	Busta aumenta di 1
8	56	Periodo aumenta di 1
9	57	Nota sale di 1

Tabella 8.2: valori ASCII, cambiamenti di tono e corrispondenti numeri sul tastierino.

corrispondenze tra tasti, codici ASCII e loro funzioni.

Il funzionamento del tasto 3 prevede che se il programma è impostato per produrre un rumore, premendo 3 passerà alla emissione di un suono musicale; se il programma è impostato per emettere un suono, emetterà allora un rumore.

```
70 ON X=48 GOSUB 120,140,160,180,200,220,240,260,280
```

Notate che nella subroutine seguente, il programma controlla i valori che sono stati cambiati, per accertarsi che rimangano entro certi limiti. L'unica eccezione si trova nella linea 160, dove la variabile WO (opzione WAVE) è passata alternativamente da 1 a 8.

```
120 O=O+1: IF O>8 THEN O=8
130 RETURN
140 O=O—1: IF O<1 THEN O=1
150 RETURN
160 IF WO=1 THEN WO=8 ELSE WO=1
170 RETURN
180 SH=SH—1: IF SH<8 THEN SH=8
190 RETURN
200 PE=PE—100: IF PE<0 THEN PE=0
```

```
210 RETURN
220 NT=NT—1: IF NT<1 THEN NT=1
230 RETURN
240 SH=SH+1: IF SH>14 THEN SH=14
250 RETURN
260 PE=PE+100: IF PE>10000 THEN PE=10000
270 RETURN
280 NT=NT+1: IF NT>12 THEN NT=12
290 RETURN
```

Ogni volta che cambiate qualcuno di questi valori premendo un tasto del tastierino, il programma aggiornerà i valori visualizzati nella parte alta dello schermo.

L'istruzione GOTOXY 0,1 colloca ciascun dato nella stessa posizione ed i nomi visualizzati sopra i numeri (grazie alla linea 40) rendono gli attuali parametri del suono facili da identificare.

```
40 GOTOXY 0,0: PRINT "Ottava:","Nota:","Forma:","Periodo:",
   "Attivatore:"
80 GOTOXY 0,1: PRINT OCT,NT,SH,PE,WO
```

Se la variabile attivatore WO è fissata al valore 8, vuol dire che dovrebbe essere emesso un suono. Nella linea 90, l'istruzione WAVE viene impiegata con gli argomenti 8 (per l'attivatore), 1 (cioè viene interessata la voce 1) e PE (per il periodo).

Una volta che il computer comincia ad emettere il suono, ritorna alla linea 50 ed aspetta eventuali altri cambiamenti.

```
90 IF WO=8 THEN WAVE 8,1,SH,PE: GOTO 50
```

Se WO è fissato al valore 1 anziché 8, il programma va alla linea 100 dove vengono impiegate sia l'istruzione WAVE che l'istruzione SOUND per produrre il rumore.

Anche qui, il programma ritorna alla linea 50 una volta che è stata emessa l'istruzione SOUND.

```
100 WAVE 1,1,SH,PE: SOUND 1,15,NT,OCT,0
110 GOTO 50
```

Infine l'inizio del programma cancella lo schermo prima che accada qualsiasi altra cosa e presenta anche i valori delle variabili PE, NT, OCT, SH e WO.

```
10 REM Programmatore dei suoni 20 FULLW 2: CLEARW 2
30 PE=1000: NT=3: OCT=3: SH=8: WO=8
40 GOTOXY 0,0: PRINT "Ottava:","Nota:","Forma:","Periodo:",
   "Attivatore:"
50 X=INP (2)
60 IF X=81 OR X=113 THEN CLEARW 2: SOUND 1,0: WAVE 0: END
70 ON X=48 GOSUB 120,140,160,180,200,220,240,260,280
80 GOTOXY 0,1: PRINT OCT,NT,SH,PE,WO
90 IF WO=8 THEN WAVE 8,1,SH,PE: GOTO 50
100 WAVE 1,1,SH,PE: SOUND 1,15,NT,OCT,0
110 GOTO 50
120 OCT=OCT+1: IF OCT)8 THEN OCT=8
130 RETURN
140 OCT=OCT-1: IF OCT(1 THEN OCT=1
150 RETURN
160 IF WO=1 THEN WO=8 ELSE WO=1
170 RETURN
180 SH=SH-1: IF SH(8 THEN SH=8
190 RETURN
200 PE=PE-100: IF PE(0 THEN PE=0
210 RETURN
220 NT=NT-1: IF NT(1 THEN NT=1
230 RETURN
240 SH=SH+1: IF SH)14 THEN SH=14
250 RETURN
260 PE=PE+100: IF PE)10000 THEN PE=10000
270 RETURN
280 NT=NT+1: IF NT)12 THEN NT=12
290 RETURN
```

Listato 8.4: programmatore dei suoni

```
10 REM Programmatore suono
20 FULLW 2: CLEARW 2
30 PE=1000: NT=3: OCT=3: SH=8: WO=8
```

Questo è un programma utile per trovare le tonalità, specialmente i rumori (dato che i suoni musicali non sono interessati dai parametri di WAVE). Il Listato 8.4 riporta il programma completo.

Miglioramenti proposti

- Aggiungete la grafica al programma in modo da visualizzare le forme d'onda selezionate.
- Mettete l'utente nella condizione di poter disporre di due o tre voci per combinare diversi toni e vederne i risultati.
- Aggiungete una funzione che permetta all'utente di variare diversi parametri (ad esempio il periodo) ad una velocità selezionabile invece di mantenerli fissi come nel programma.

GRAFICI A ISTOGRAMMI E LINEARI

All'inizio, in questo capitolo, abbiamo costruito un programma che disegnava un diagramma circolare in base ai dati che una persona inseriva dalla tastiera.

Anche quest'ultimo programma disegna grafici, ad istogrammi e lineari, ma la programmazione è un po' differente.

Descrizione del programma

Sviluppate un programma che accetti fino a 100 dati numerici dalla tastiera per tracciare da queste informazioni un grafico ad istogrammi o lineare (secondo la scelta dell'utente).

Se si disegna un grafico ad istogrammi, essi dovranno essere tutti della stessa larghezza, il numero più alto dovrebbe terminare nella parte alta dello schermo e tutti gli altri numeri dovrebbero essere di dimensioni proporzionali; gli istogrammi vanno inoltre separati in modo che non si tocchino. Cercate di distinguere gli istogrammi con colori o colorazioni di fondo diverse.

Se l'utente seleziona il grafico lineare, fate in modo che il punto più alto del grafico si trovi verso la sommità dello schermo ed assicuratevi che tutti i punti sul grafico siano orizzontalmente equidistanti.

Componenti

1. Usate una matrice per registrare i dati che l'utente inserisce.
2. Mentre i dati vengono inseriti, usate una variabile chiamata BIG per registare il numero più grande fino a quel momento inserito. Se il dato appena inserito è maggiore di BIG, allora ponete BIG uguale al numero appena inserito (quando il programma inizia, BIG è uguale a 0 e il primo numero verrà registrato automaticamente in BIG).
   Usando questo semplice ragionamento sarete certi che BIG sia il numero inserito più grande e conoscere il numero più grande è importante perché vi permette di proporzionare le altezze degli istogrammi o dei punti del grafico lineare.
3. Usate il numero di dati inseriti per stabilire la larghezza degli istogrammi. Inoltre, quando disegnate gli istogrammi, interponete lo spazio di qualche pixel tra l'uno e l'altro, indipendentemente dalle dimensioni degli istogrammi.

Sviluppo del programma

L'utente dovrà innanzitutto inserire i dati. Mentre i dati vengono inseriti, essi vengono registrati nella matrice D(). La variabile PD conta quanti dati sono stati inseriti e OYC è una variabile arbitraria per il grafico lineare che registra la vecchia ordinata.

```
10 REM Grafici a istogrammi e lineari
20 FULLW 2: CLEARW 2
30 DIM D(100)
40 PD = 1:OYC = 150
```

Per ciascun dato inserito, il computer verifica se il suo valore numerico coincide con due condizioni speciali. Se D(PD), il numero appena inserito, è uguale a 333, il computer cancellerà lo schermo e finirà il programma, poiché questo numero segnala che l'utente vuole uscirne. Se D(PD) non è uguale a 333, il computer verifica se è uguale a 0, che significa che l'utente ha finito di battere i numeri; se viene superato questo test logico, il computer sottrae 1 da PD (poiché lo 0 inserito come dato è inutile e non va contato come punto del diagramma) e il programma chiede se l'utente vuole stampare un grafico ad istogrammi o un grafico lineare. La risposta verrà registrata in GCH e verrà usata più avanti nel programma.

```
50 PRINT "Inserisci il dato ";PD;: INPUT D(PD)
60 IF D(PD) = 333 THEN CLEARW 2: END
70 IF D(PD) = 0 THEN PD = PD—1: PRINT: INPUT "Grafico a
   istogrammi (1) o lineare (2)";GCH: GOTO 100
```

Se D(PD) non è né 333 né 0, il computer presume che sia stato inserito un nuovo dato e lo confronta con la variabile BIG. BIG permette al computer di registrare il dato più grande fino a quel momento inserito. Se il dato inserito è maggiore di BIG, allora BIG viene posto uguale a D(PD). Dopo che il computer ha ricevuto una serie di dati, incrementa la variabile PD di 1 e ritorna alla linea 50 per ricevere altre serie di dati.

```
80 IF D(PD)>BIG THEN BIG = D(PD)
90 PD = PD + 1: GOTO 50
```

La routine che disegna il grafico è piuttosto semplice, ma se cercate di stendere il programma per conto vostro, potreste trovarvi in difficoltà a decidere come comportarvi di fronte a tutti i tipi di numeri che l'utente può inserire. Dopo che lo schermo è stato cancellato, il computer deve determinare un mol-

tiplicatore per tutti i punti. Questo moltiplicatore assicurerà che il grafico ad istogrammi non supererà i bordi dello schermo, e neppure sarà piccolo al punto da renderne poco agevole la lettura. Questo moltiplicatore, la variabile IN, si calcola dividendo 120 (il numero di pixel verticali che ho riservato al grafico) per BIG. Supponiamo che il numero più grande inserito sia 480; 120 diviso per 480 dà 0.25 e quindi ogni dato verrà moltiplicato per questo numero nel loop FOR...NEXT della linea 110. Il numero 480 verrà ridotto in scala a 120 (l'altezza massima che avete fissato per gli istogrammi o per i punti della linea) e tutti gli altri numeri saranno proporzionatamente più piccoli. Questa tecnica è semplice, ma risolve un sacco di problemi nella costruzione del grafico.

```
100 CLEARW 2
110 IN—120/BIG: FOR I=1 TO PD: D(I)=130—(IN*D(I)): NEXT
```

Indipendentemente da quale grafico stia per disegnare, il programma, ora traccia le due linee per gli assi delle ascisse e delle ordinate.

```
120 LINEF 20,150,600,150: LINEF 20,150,20,20
```

Per calcolare la larghezza degli istogrammi, o la distanza tra i punti del grafico lineare, il programma divide 450 per il numero dei dati numerici inseriti (variabile PD). Si usa il numero 450 perché quello è il numero dei pixel misurati orizzontalmente sullo schermo. Quindi, se ci sono 25 punti, la variabile larghezza degli istogrammi (BW) sarà uguale a 18, il che significa che gli istogrammi saranno larghi 18 pixel ed i punti distanti 23 pixel l'uno dall'altro (capirete tra poco la differenza di 5 pixel).

```
130 BW=450/PD
```

A questo punto il computer può disegnare il grafico. Il loop FOR...NEXT conta da 30 a 30+((BW+5)*PD) STEP BW+5. Prendiamo in esame ciascuna di queste parti separatamente: il loop inizia a 30 perché quella è la collocazione dell'asse delle ascisse. Il valore STEP di BW+5 assicura che ci saranno 5 pixel di spazio tra un istogramma e l'altro; se invece si tratta di un grafico lineare, la distanza tra i punti sarà semplicemente di BW+5. Il loop conta fino a 30+((BW+5)*PD) poiché 30 è il punto di partenza del loop, BW+5 è il numero di pixel alla cui destra verrà tracciato il successivo istogramma e PD è il numero dei dati che verranno visualizzati.

```
140 FOR X=30 TO 30+((BW+5)*PD) STEP BW+5
```

Per chiarire le operazioni di questo loop, supponiamo che aveste cinque dati: 32, 54, 44, 48 e 62. Siccome ci sono 5 numeri, la variabile BW (uguale a 450/PD) sarà uguale a 90, che significa che ogni istogramma sarà largo 90 pixel. Il loop quindi conterà da 30 a 505 con incrementi di 95, stando ad indicare che il lato sinistro degli istogrammi si troverà a 30, 125, 220, 315, 410 e 505. La posizione dei punti nel caso del grafico lineare sarà uguale a questi stessi valori.

Il programma ora incrementa la variabile contatore C di 1 e la confronta con PD. Se il contatore è maggiore del numero dei dati, il programma ha finito di disegnare il grafico ed andrà alla linea 200 dove è impiegata la funzione INP per vedere se è stata premuta una G maiuscola o minuscola. Se tale tasto è stato premuto, il computer presumerà che l'utente voglia un altro grafico e ritornerà alla linea 30. Se è stato premuto qualsiasi altro tasto, il computer si limiterà a lasciare il grafico sullo schermo e attenderà indefinitamente.

```
150 C=C+1: IF C>PD THEN GOTO 200
200 IF INP(2)=ASC("g") OR INP(2)=ASC("G") THEN CLEARW 2:
    CLEAR: GOTO 30
210 GOTO 200
```

Se GCH è uguale a 2 (che significa che l'utente vuole un grafico lineare), il computer disegna una linea dal pixel (X,OYC) a (X+BW+5,D(C)). X è la variabile FOR...NEXT, OYC è la vecchia ordinata, X+BW+5 è il punto successivo dove il loop procederà e D(C) è il punto che il computer ha calcolato prima. Notate che OYC è posto uguale a D(C) dopo che la linea è stata tracciata, in modo che il computer possa ricordare in che punto si è interrotta la linea quando deve essere tracciata la linea successiva. Il GOTO 190 fa saltare il programma all'istruzione NEXT di quella linea in modo che il loop possa continuare fino alla fine.

```
160 IF GCH=2 THEN LINEF X,OYC,X+BW+5,D(C): OYC=D(C):
    GOTO 190
```

Se GCH non è uguale a 1, il computer disegnerà l'istogramma corrispondente ai dati inseriti. Il lato sinistro dell'istogramma va da (X,150) che si trova sull'asse delle ascisse, a (X,D(C)), cioè al punto determinato dal lato inserito. La linea che forma il lato superiore dell'istogramma va da (X,D(C)) a (X+BW,D(C)): l'ordinata resta la stessa, dato che vogliamo che la parte superiore dell'istogramma sia piatta, ma la seconda ascissa viene incrementata di BW, che è la larghezza degli istogrammi in pixel. Il lato destro dell'istogramma va da (X+BW,D(C)) a (X+BW,150). Esaminate attentamente questa linea per vedere come sia il contatore X del loop FOR...NEXT sia la larghezza dell'istogramma BW sono impiegati per sempificare il disegno dell'istogramma.

```
10 REM Grafici a istogrammi e lineari
20 FULLW 2: CLEARW 2
30 DIM D(100)
40 PD=1:OYC=150
50 PRINT "Inserisci il dato ";PD;: INPUT D(PD)
60 IF D(PD)=333 THEN CLEARW 2: END
70 IF D(PD)=0 THEN PD=PD-1: PRINT: INPUT "Grafico a istogrammi (1)
   o lineare (2)";GCM: GOTO 100
80 IF D(PD)>BIG THEN BIG=D(PD)
90 PD=PD+1: GOTO 50
100 CLEARW 2
110 IN=120/BIG: FOR I=1 TO PD: D(I)=130-(IN*D(I)): NEXT
120 LINEF 20,150,600,150: LINEF 20,150,20,20
130 BW=450/PD
140 FOR X=30 TO 30+((BW+5)*PD) STEP BW+5
150 C=C+1: IF C>PD THEN GOTO 200
160 IF GCM=2 THEN LINEF X,OYC,X+BW+5,D(C): OYC=D(C): GOTO 190
170 LINEF X,150,X,D(C): LINEF X,D(C),X+BW,D(C): LINEF
    X+BW,D(C),X+BW,150
180 COLOR 1,3,1,C,3: FILL X+1,149
190 NEXT
200 IF INP(2)=ASC("g") OR INP(2)=ASC("G") THEN CLEARW 2: CLEAR:
    GOTO 30
210 GOTO 200
RUN
Grafico a istogrammi (1) o lineare (2)? 1
Inserisci il dato 1?  45
Inserisci il dato 2?  65
Inserisci il dato 3?  13
Inserisci il dato 4?  145
Inserisci il dato 5?  76
Inserisci il dato 6?  54
Inserisci il dato 7?  92
Inserisci il dato 8?  0
```

Listato 8.5: programma Grafici a istogrammi e lineari

```
170 LINEF X,150,X,D(C): LINEF X,D(C),X + BW,D(C): LINEF X + BW,
    D(C),X + BW,150
```

Tutte le volte che viene tracciato un istogramma, le istruzioni COLOR e FILL vengono impiegate nella linea 180 per riempirlo di un motivo casuale. Questo permette di distinguere ancora meglio gli istogrammi. Infine la linea 190 chiude il loop.

```
180 COLOR 1,3,1,C,3: FILL X + 1,149
190 NEXT
```

Il Listato 8.5 presenta tutte le linee di programma in ordine, insieme ad un esempio dell'input. La Figura 8.3 mostra il grafico ad istogrammi di questi dati e la Figura 8.4 mostra l'equivalente grafico lineare.

Miglioramenti proposti

- Mettete l'utente in grado d'inserire un titolo per il grafico e di etichettare il periodo o un'altra quantità rappresentata dagli istogrammi o dai punti del grafico.

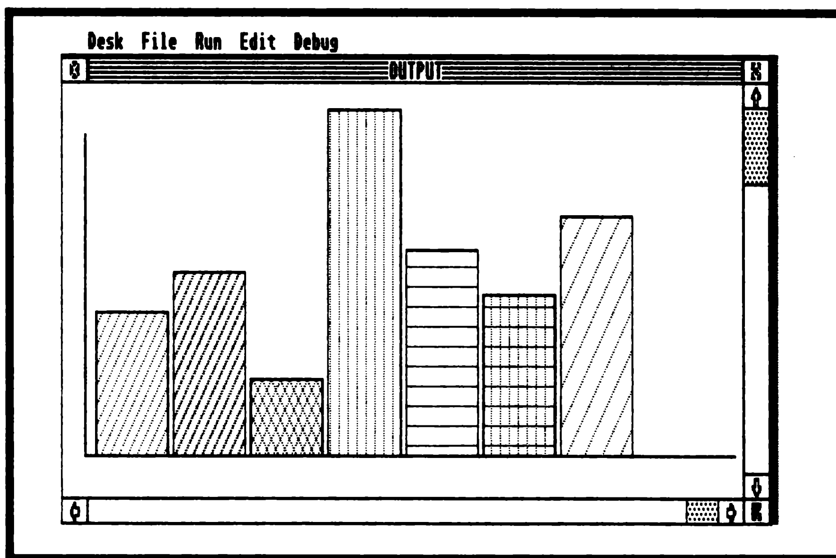


Figura 8.3: un grafico ad istogrammi dei dati impiegati nell'esecuzione esemplificativa.

- In base alla riduzione in scala, tarate l'asse delle ordinate da zero al valore massimo rappresentato.
- Per un progetto veramente ambizioso, combinate il programma dei diagrammi circolari con questo in modo che l'utente possa scegliere tra tre tipi di grafico.
- Aggiungete una routine d'input in modo che il programma suggerisca quale tipo di grafico è il più appropriato alla natura dei dati inseriti.

SUGGERIMENTI PER LA PROGRAMMAZIONE

Indipendentemente da come abbiate scritto questi programmi, vediamo alcuni criteri generali che vi potranno aiutare quando scriverete i vostri programmi in base a vostre idee.

- Prestate molta attenzione ai nomi delle variabili, dato che è facile confonderle. Per esempio, se state usando una variabile chiamata PF e passate accidentalmente a FP, probabilmente avrete un errore nel programma. Il modo più facile di evitare questo problema è di dare alle variabile nomi facili da ricordare, in stretta relazione con la finalità della variabile.

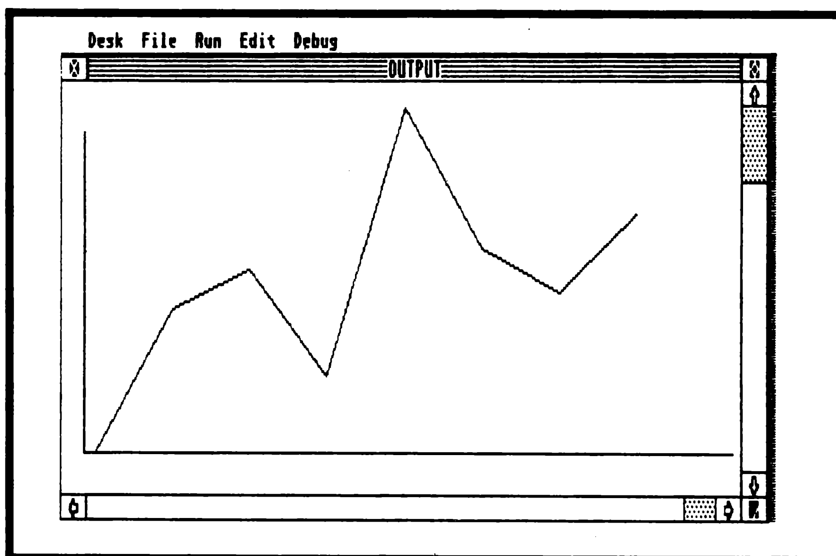


Figura 8.4: un grafico lineare in base agli stessi dati.

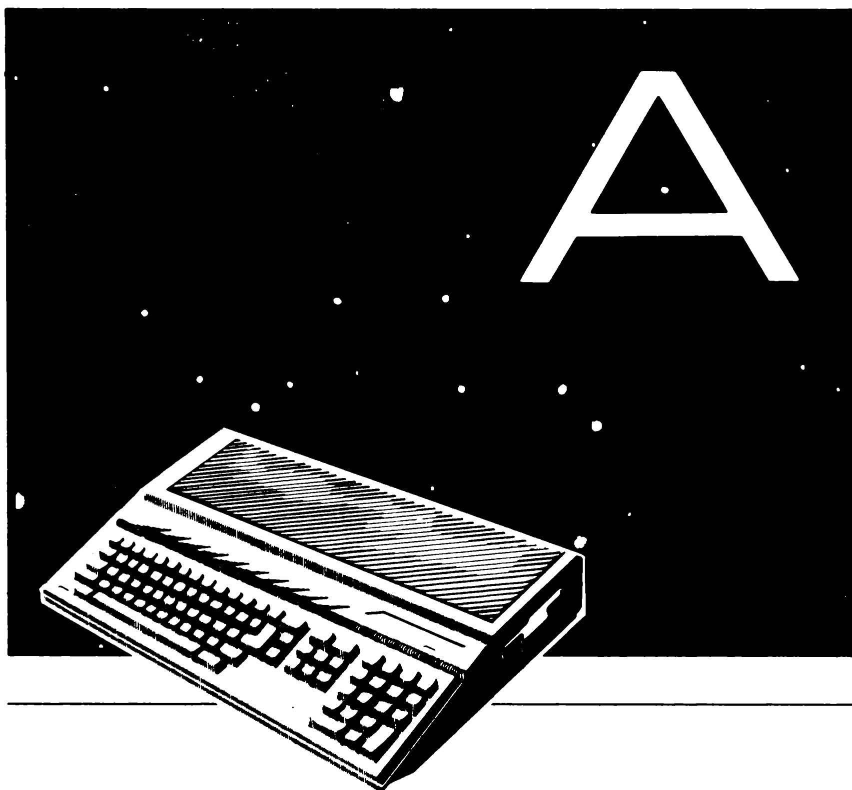
- Non cercate di scrivere il programma dalla linea 1 alla fine senza saltare qua e là alle diverse sezioni. Più verosimilmente scriverete prima le sezioni chiave del programma ed in base a quelle tutto il resto. Scrivere un programma vuol dire procedere a costruire, migliorare e depurare dai bug. E, come nell'arte, un programma non può mai essere finito, ma solo abbandonato. Ci saranno sempre altre funzioni che potreste aggiungere; quando il programma è ad un livello sufficiente a svolgere il lavoro che vi siete proposti ed è facile da capire e da usare, la vostra missione si può dire compiuta.
- Anche se molti autori suggeriscono diagrammi di flusso, algoritmi e molta progettazione prima d'iniziare un programma, non ci sono molti programmatori di personal computer che effettivamente procedano secondo questi stadi di preparazione.
  Per la maggior parte la gente si siede con un'idea generale di ciò che vuole e inizia a battere. I semplici stadi di progettazione che abbiamo visto illustrati in questo capitolo sono di solito sufficienti. Un sacco di idee e di soluzioni migliori emergono durante la fase di rifinitura del programma e qualche volta i bug vengono lasciati perché producono nuovi effetti interessanti. La parola magica della programmazione è semplicità e purché abbiate familiarità con il linguaggio, vi divertirete ed imparerete molto di più partendo da un'idea e procedendo con quella.

LAVORARE E PROGRAMMARE CON L'ST

In questo libro abbiamo trattato insieme un terreno molto ampio; ormai dovreste essere in grado di scrivere un buon programma in BASIC ST, sapete cosa sta dietro alla grafica e alla sintesi acustica, conoscete alcune applicazioni del vostro ST e potete sfidare voi stessi a realizzare progetti interessanti con il vosto computer.

- L'esperienza è l'insegnante migliore. Non lasciatevi prendere da un eccesso di sfiducia tutte le volte che trovate un bug nel programma, anzi capirete che inciampando qua e là ed imparando le vostre lezioni di programmazione alla maniera dura, sarete programmatori migliori che se aveste letto dieci libri di programmazione e mai toccato la tastiera. Questo libro è stato scritto per darvi la spinta iniziale, ma per la maggior parte le migliori lezioni saranno quelle da autodidatta. Se proprio volete ancora un aiuto esterno, le risorse migliori sono i gruppi di utenti degli ST, le riviste che contengono informazioni sui computer Atari ed alcuni libri sui computer attentamente selezionati.
- Aspettate qualche mese dopo l'uscita di qualche prodotto strepitoso per il vostro computer. Per quanto irresistibilmente siate tentati di correre a comperare ogni programma innovativo o tutte le periferiche che compaiono, statevene tranquilli per un pò ed aspettate che il prodotto dimostri la propria qualità e che il prezzo scenda un po'.
- Fate sempre una copia dei vostri lavori, tutte quelle che potete il più spesso possibile! Mi è capitato di sprecare interi giorni di lavoro per non aver fatto delle copie frequenti. Ricordate che la potenza che il computer può mettere a vostra disposizione può altrettanto facilmente rivolgersi contro di voi. Il vostro ST può essere un amico o un altrettanto potente nemico, secondo l'accortezza con cui lo usate.

Pare proprio che l'ST abbia un futuro davanti a sé. Nei prossimi anni verranno messi sul mercato nuovi pacchetti software e periferiche avanzate. Se saprete programmare il vostro Atari e se capirete quanto può essere utile un personal computer, potrete apprezzare il vostro ST molto di più e capire tutte le grandi cose che con esso la gente, voi compresi, può realizzare.



LE PAROLE CHIAVE DEL BASIC ST

Quello che segue è un elenco in ordine alfabetico delle parole chiave del BASIC ST, dei loro formati e del capitolo in cui potete trovarne esempi e dettagliata descrizione. Ogni parola chiave non identificata come comando o funzione, è un'istruzione.

ABS (Capitolo 3)

Formato:

variabile numerica =ABS(*espressione numerica*)

Finalità: dà il valore assoluto di un numero; funzione.

AND (Capitolo 2)

Formato:

espressione1 AND *espressione2*

Finalità: verifica logicamente se due espressioni sono vere.

ASC (Capitolo 3)

Formato:

variabile intera =ASC(*espressione di stringa*)

Finalità: dà il valore ASCII di un carattere; funzione.

ATN (Capitolo 3)

Formato:

variabile numerica =ATN(*espressione numerica*)

Finalità: dà l'arcotangente di un numero, misurato in radianti; funzione.

AUTO (Capitolo 2)

Formato:

AUTO *inizio,incremento*

Finalità: numera automaticamente le linee di un programma; comando diretto.

BREAK (Capitolo 2)

Formato:

BREAK *linea[,linea,...]*

Finalità: stampa linee specifiche sullo schermo man mano che il programma le incontra; comando diretto. Quando il programma incontra una linea che voi avete specificato, si ferma e visualizza il numero della linea; per far ripartire il programma, basta battere CONT o premere il tasto Return.

CHAIN (Capitolo 7)

Formati:

CHAIN *"nome del file"[,numero di linea,ALL]*
CHAIN MERGE *"nome del file",linea,DELETE numero di linea[-numero di linea]*

Finalità: permette ad un programma di caricare e far girare un altro programma. Nel primo formato, le variabili del programma in memoria possono essere conservate nel procedimento. Nel secondo formato, il programma caricato viene unito al programma in memoria e certe linee del vecchio programma possono essere cancellate. Entrambi i formati vi permettono di specificare il numero di linea iniziale del nuovo programma.

CHR$ (Capitolo 3)

Formato:

variabile di stringa =CHR$*(espressione numerica)*

Finalità: converte un numero ASCII nel carattere corrispondente; funzione.

CINT (Capitolo 3)

Formato:

variabile numerica =CINT*(espressione numerica)*

Finalità: arrotonda un numero all'intero più vicino; funzione.

CIRCLE (Capitolo 4)

Formato:

CIRCLE *X,Y,raggio[,angolo iniziale,angolo finale]*

Finalità: disegna un cerchio con centro (X,Y) e con gli argomenti che avrete inserito; gli angoli sono misurati in decimi di grado.

CLEAR (Capitolo 3)

Formato:

CLEAR

Finalità: cancella le variabili, i contatori e le matrici dalla memoria; comando diretto.

CLEARW (Capitolo 2)

Formato:

CLEARW *numero della finestra*

Finalità: cancella il contenuto della finestra da voi specificata; i numeri delle finestre sono 0 (Edit), 1 (List), 2 (Output) e 3 (Command).

CLOSE (Capitolo 7)

Formato:

CLOSE *[#]numero[,numero,...]*

Finalità: chiude i file specificati tramite il numero.

CLOSEW (Capitolo 2)

Formato:

CLOSEW *numero della finestra*

Finalità: toglie una finestra dallo schermo; 0 (Edit), 1 (List), 2 (Output) o 3 (Command).

COLOR (Capitolo 4)

Formato:

COLOR *[colore del testo,colorazione del fondo,colore della linea,stile,indice]*

Finalità: imposta il colore e lo stile del testo e la colorazione del fondo; almeno un argomento è necessario, secondo il contesto.

COMMON (Capitolo 7)

Formato:

COMMON *variabile[,variabile,...]*

Finalità: vi permette di specificare quali variabili devono essere conservate quando usate l'istruzione CHAIN. Senza COMMON, tutte le variabili possono essere conservate.

CONT

Formato:

CONT

Finalità: permette al programma di continuare dal punto in cui era stato interrotto da un comando STOP o BREAK; comando diretto.

CONTRL (Capitolo 6)

Formato:

CONTRL*(numero)=valore*
POKE CONTRL+*numero*2,valore*

Finalità: nome di variabile usata insieme alle routine VDISYS; rappresenta la cella della matrice di controllo VDI del GEM.

COS (Capitolo 3)

Formato:

variabile numerica=COS(espressione numerica)

Finalità: dà il coseno di un angolo misurato in radianti; funzione.

CVD, CVI, CVS (Capitolo 7)

Formato:

CVD*(stringa di otto caratteri)*
CVI*(stringa di due caratteri)*
CVS*(stringa di quattro caratteri)*

Finalità: ciascuna di queste funzioni converte una stringa in un numero con cui un file ad accesso casuale può lavorare.

DATA (Capitolo 2)

Formato:

DATA *numero o stringa[,numero o stringa,...]*

Finalità: contiene il numero o la stringa necessari ad un'istruzione READ.

DEFDBL, DEFINT, DEFSNG, DEFSTR (Capitolo 3)

Formato:

DEFDBL *lettera[-lettera]*
DEFINT *lettera[-lettera]*
DEFSNG *lettera[-lettera]*
DEFSTR *lettera[-lettera]*

Finalità: definisce il nome di una variabile rispettivamente come a precisione doppia, intero, a precisione singola o stringa. Potete anche specificare un intervallo di nomi di variabile, DEFINT A-C rendendo A, B e C variabili intere.

DEF FN (Capitolo 3)

Formato:

DEF FN*FUNCTION NAME(variabile[,variabile,...])=espressione*

Finalità: definisce una funzione sotto un particolare nome usando certe variabili.

DEF SEG (Capitolo 6)

Formato:

DEF SEG *espressione numerica*

Finalità: stabilisce se le istruzioni PEEK o POKE operano con i byte (DEF SEF 0) oppure con parole di due byte (DEF SEG 1).

DELETE (Capitolo 2)

Formato:

DELETE *[numero-numero]*

Finalità: toglie linee di programma dalla memoria; si specifica un numero o un intervallo; comando diretto.

DIM (Capitolo 2)

Formato:

DIM *variabile(indice[,indice,...])*

Finalità: riserva una certa quantità di memoria per una matrice di variabili di qualsiasi dimensione.

DIR (Capitolo 7)

Formato:

DIR*[drive:stringa di ricerca]*

Finalità: visualizza il contenuto della directory di un disco; comando diretto.

EDIT (Capitolo 2)

Formato:

EDIT *[numero di linea]*

Finalità: mette il BASIC ST nella finestra Edit in modo che possiate modificare il programma in memoria; comando diretto.

ELLIPSE (Capitolo 4)

Formato:

ELLIPSE *X,Y,raggio orizzontale,raggio verticale[,angolo iniziale,angolo finale]*

Finalità: disegna un'ellisse con centro (X,Y) e con gli altri argomenti da voi specificati.

ELSE (Capitolo 2)

Formato:

IF *condizione* THEN *azione1* ELSE *azione2*

Finalità: parte di un'istruzione IF...THEN; fa eseguire al computer un'azione alternativa se la condizione da verificare non è vera.

END (Capitolo 2)

Formato:

END

Finalità: ferma il programma e vi riporta alla finestra Command.

EOF (Capitolo 7)

Formato:

EOF*(numero del file)*

Finalità: funzione; dà 0 finché non legge l'ultima voce di un file sequenziale, poi dà —1.

EQV (Capitolo 2)

Formato:

espressione EQV *espressione*

Finalità: test logico per vedere se due espressioni sono entrambe vere o entrambe false.

ERA (Capitolo 7)

Formato:

ERA *[drive:]nome del file*

Finalità: cancella un file dal disco.

ERASE (Capitolo 3)

Formato:

ERASE *nome della matrice[,nome della matrice,...]*

Finalità: cancella una o più matrici dalla memoria.

ERL (Capitolo 2)

Formato:

ERL

Finalità: variabile; memorizza il numero della linea dove si è verificato l'errore più recente.

ERR (Capitolo 2)

Formato:

ERR

Finalità: variabile; memorizza il più recente codice di errore.

ERROR (Capitolo 2)

Formato:

ERROR *codice di errore*

Finalità: simula un errore; comando diretto.

EXP (Capitolo 3)

Formato:

variabile numerica = EXP*(espressione numerica)*

Finalità: dà il valore di *e* (2.71828) elevato alla potenza specificata; funzione.

FIELD (Capitolo 7)

Formato:

FIELD *#numero,byte* AS *variabile di stringa[,bytes* AS *variabile di stringa]*

Finalità: riserva un certo numero di byte in memoria per variabili che verranno usate in un file ad accesso diretto.

FILL (Capitolo 4)

Formato:

FILL *X,Y*

Finalità: riempie un'area usando il colore e la colorazione del fondo impostata dalla più recente istruzione COLOR, a cominciare dal pixel (X,Y) per finire lungo il bordo circostante.

FIX (Capitolo 3)

Formato:

variabile numerica = FIX*(espressione numerica)*

Finalità: elimina la parte decimale di un numero; funzione.

FOLLOW (Capitolo 2)

Formato:

FOLLOW *variabile[,variabile,...]*

Finalità: fa visualizzare al computer i valori della o delle variabili che specificate mentre un programma sta girando; comando diretto.

FOR (Capitolo 2)

Formato:

FOR *indice=inizio* TO *fine [*STEP *incremento]*
corpo del loop
NEXT

Finalità: la prima parte di un loop FOR...NEXT.

FULLW (Capitolo 4)

Formato:

FULLW *numero della finestra*

Finalità: allarga la finestra che specificate all'intero schermo.

GEMSYS (Capitolo 6)

Formato:

GEMSYS*(opcode)*

Finalità: funzione; inizia una funzione AES.

GET (Capitolo 7)

Formato:

GET *#numero, numero del record*

Finalità: recupera un record da un file ad accesso diretto.

GOSUB (Capitolo 2)

Formato:

GOSUB *numero di linea o etichetta[,numero di linea o etichetta]*

Finalità: trasferisce il controllo ad una certa o a certe linee del programma; quando incontra un RETURN, il computer torna al punto del programma immediatamente seguente a GOSUB.

GOTO (Capitolo 2)

Formato:

GOTO *numero di linea o etichetta*

Finalità: trasferisce l'esecuzione del programma ad un certo numero di linea nel programma.

GOTOXY (Capitolo 3)

Formato:

GOTOXY *colonna,riga*

Finalità: posiziona il cursore del testo alla colonna e alla riga dello schermo specificate.

HEX$ (Capitolo 3)

Formato:

variabile numerica = HEX$(*espressione numerica*)

Finalità: converte un numero decimale in esadecimale (base 16); funzione.

IF (Capitolo 2)

Formato:

IF *condizione* THEN *azione1* [ELSE *azione2*]

Finalità: prima parte di un'istruzione IF...THEN; immediatamente seguita da un test logico.

INP (Capitolo 7)

Formato:

INP *(porta)*

Finalità: funzione; riceve un byte d'informazione dalla porta che specificate; stampante (0), porta RS232 (1), tastiera (2), MIDI (3).

INPUT (Capitolo 2)

Formato:

INPUT *["prompt"];variabile[,variabile,...]*

Finalità: riceve dati numerici o di stringa dalla tastiera; potete usare un prompt per dire all'utente quali informazioni servono, oppure potete tralasciarlo; un inserimento viene terminato da un RETURN.

INPUT (Capitolo 7)

Formato:

INPUT *#numero,variabile[,variabile,...]*

Finalità: carica una linea di dati dal drive; deve esserci un file sequenziale aperto perché questa istruzione operi.

INSTR (Capitolo 3)

Formato:

INSTR *([carattere iniziale,]stringa da cercare, escursione di ricerca)*

Finalità: funzione; dà la posizione iniziale del segmento che volete trovare nell'ambito di una certa stringa presa in esame. Se specificate una posizione iniziale, il computer inizierà a cercare quella posizione, nell'ambito della stringa per trovare il vostro segmento.

INT (Capitolo 3)

Formato:

variabile numerica = INT*(espressione numerica)*

Finalità: arrotonda un numero all'intero più vicino; funzione.

INTIN (Capitolo 6)

Formato:

INTIN*(numero)* = *valore*
POKE INTIN + *numero*2,valore*

Finalità: variabile per uso con le funzioni VDISYS; permanentemente conservata in memoria.

INTOUT (Capitolo 7)

Formato:

INTOUT*(numero)*
PEEK (INTOUT+ *numero*2)*

Finalità: variabile usata con VDISYS per recuperare un byte di dati anziché memorizzarlo.

KILL (Capitolo 7)

Formato:

KILL *"stringa" o espressione di stringa*

Finalità: toglie un file da un disco; KILL è più versatile di ERA perché si può usare KILL all'interno di un programma.

LEFT$ (Capitolo 3)

Formato:

stringa = LEFT$*(stringa madre,numero dei caratteri)*

Finalità: funzione; estrae un certo numero di caratteri dalla sinistra di una stringa. Solo la stringa madre rimane inalterata.

LEN (Capitolo 3)

Formato:

intero = LEN *(espressione di stringa)*

Finalità: fornisce il numero di caratteri di una stringa; funzione.

LET (Capitolo 2)

Formato:

LET *nome della variabile = espressione*

Finalità: assegna un valore o una stringa ad un nome di variabile; LET non si usa quasi mai, poiché potete battere direttamente *nome della variabile* = *espressione* ottenendo lo stesso risultato.

LINEF (Capitolo 4)

Formato:

LINEF *X1, Y1, X2, Y2*

Finalità: disegna una linea dal punto (X1,Y1) al punto (X2,Y2).

LINE INPUT (Capitolo 2)

Formato:

LINE INPUT *["prompt";]variabile di stringa*

Finalità: vi permette d'inserire una stringa d'informazioni con le virgolette, virgole e tutti i segni di punteggiatura che normalmente non sono permessi con INPUT.

LINE INPUT (Capitolo 7)

Formato:

LINE INPUT *#numero,variabile di stringa*

Finalità: come INPUT#, questo comando ottiene informazioni direttamente dal drive; tuttavia, gli argomenti possono contenere caratteri di solito non permessi con INPUT#, come le virgole, le virgolette e i punto e virgola.

LIST (Capitolo 2)

Formato:

LIST *[numero di linea-numero di linea]*

Finalità: visualizza tutto o parte del programma attualmente in memoria; comando diretto.

LLIST (Capitolo 7)

Formato:

LLIST *[numero di linea-numero di linea]*

Finalità: invia alla stampante tutto o parte del programma attualmente in memoria.

LOAD (Capitolo 2)

Formato:

LOAD *nome del file*

Finalità: prende un programma dal drive e lo carica in memoria.

LOC (Capitolo 7)

Formato:

LOC*(numero del file)*

Finalità: funzione; dà o il numero dell'ultimo record usato se si sta usando un file ad accesso casuale oppure il numero di record usati da quando si è aperto il file se si sta usando un file sequenziale.

LOF (Capitolo 7)

Formato:

LOF *(#numero del file)*

Finalità: dà la lunghezza di un file misurata in byte; funzione.

LOG (Capitolo 3)

Formato:

variabile numerica = LOG*(espressione numerica)*

Finalità: dà il logaritmo naturale di un numero; funzione.

LOG10 (Capitolo 3)

Formato:

variabile numerica = LOG10*(espressione numerica)*

Finalità: dà il logaritmo in base 10 di un numero; funzione.

LPOS (Capitolo 7)

Formato:

LPOS (0)

Finalità: determina la posizione attuale della testina della stampante; funzione.

LPRINT (Capitolo 7)

Formato:

LPRINT *espressione*

Finalità: stampa una stringa, un numero o altre espressioni sulla stampante.

LSET (Capitolo 7)

Formato:

LSET *nuova stringa=vecchia stringa*

Finalità: prepara una stringa da salvare in un file ad accesso diretto sotto il nome *nuova stringa.*

MERGE (Capitolo 7)

Formato:

MERGE *nome del file*

Finalità: carica un file dal disco e lo combina con il file in memoria; comando diretto.

MID$ (Capitolo 3)

Formato:

variabile di stringa =MID$(*stringa,punto iniziale,lunghezza*)
MID$(*stringa,punto iniziale,lunghezza*)=*variabile di stringa*

Finalità: funzione; il primo formato trova un segmento di stringa di una certa lunghezza che inizia con un certo carattere entro la stringa. Questo segmento viene memorizzato in una variabile specifica. Il secondo formato memorizza

un nuovo segmento di stringa entro la stringa che inizia ad un certo carattere e che continua per un numero di caratteri pari a *lunghezza.*

MKD$, MKI$, MKS$ (Capitolo 7)

Formato:

MKD$*(valore numerico)*
MKI$*(valore numerico)*
MKS$*(valore numerico)*

Finalità: quando convertite una stringa in un numero con CVD, CVI e CVS, una di queste funzioni deve essere impiegata per riconvertire quel numero in una stringa; usate la funzione corrispondente al tipo di valore che viene caricato dal file ad accesso casuale; userete MKD$ per i numeri a precisione doppia, MKS$ per i numeri a precisione singola e MKI$ per gli interi.

MOD (Capitolo 3)

Formato:

variabile numerica=numero MOD *divisore*

Finalità: dà il resto della divisione di *numero* per il *divisore.*

NAME (Capitolo 7)

Formato:

NAME *vecchio nome* AS *nuovo nome*

Finalità: cambia il nome del file da *vecchio nome* a *nuovo nome.*

NEW (Capitolo 2)

Formato:

NEW

Finalità: toglie il programma e le variabili dalla memoria (cancella l'intera area di lavoro del BASIC); comando diretto.

NEXT (Capitolo 2)

Formato:

NEXT

Finalità: chiude un loop FOR...NEXT; quando incontra NEXT, il computer ricomincia il loop.

NOT (Capitolo 2)

Formato:

NOT *espressione*

Finalità: questa espressione logica inverte il risultato di qualsiasi test logico. Vero diventa falso e falso diventa vero.

OCT$ (Capitolo 3)

Formato:

variabile numerica =OCT$(*espressione numerica*)

Finalità: converte un numero decimale in ottale (base 8); funzione.

ON (Capitolo 2)

Formato:

ON *variabile* GOTO *linea o etichetta[,linea o etichetta,...]*
ON *variabile* GOSUB *linea o etichetta[,linea o etichetta,...]*
ON ERROR GOTO *linea o etichetta*

Finalità: fa andare il computer al numero di linea corrispondente al valore

della variabile; se la variabile è uguale a 1, il computer va al primo numero di linea specificato, se la variabile è uguale a 2, il computer va al secondo numero di linea e così via.

Sia GOTO che GOSUB funzionano con ON e ON ERROR GOTO fa andare il computer al numero di linea specificato se incontra un errore da qualsiasi parte nel programma.

OPEN (Capitolo 7)

Formato:

OPEN *"tipo", #numero,"nome del file"[,lunghezza]*

Finalità: apre un file in modo che possa essere usato per record ad accesso diretto o sequenziali; il tipo è "O" per l'output di file sequenziali e "I" per l'input di file sequenziali, oppure "R" per input/output di file ad accesso diretto; la lunghezza standard di un file è di 128 byte.

OPENW (Capitolo 2)

Formato:

OPENW *numero di finestra*

Finalità: fa comparire una finestra sullo schermo, sovrapponendola a tutte le altre finestre; le finestre sono 0 (Edit), 1 (List), 2 (Output) e 3 (Command).

OPTION BASE (Capitolo 2)

Formato:

OPTION BASE 0 o 1

Finalità: stabilisce il primo elemento (indice) per le matrici di variabili; OPTION BASE 0 fissa H(0) come primo elemento della matrice H() e OPTION BASE 1 fissa H(1) come primo elemento della stessa matrice.

OR (Capitolo 2)

Formato:

espressione1 OR *espressione2*

Finalità: questo test logico dà il valore *vero* se una o entrambe le espressione sono vere.

OUT (Capitolo 7)

Formato:

OUT *porta,byte*

Finalità: manda un byte alla porta specificata.

PCIRCLE (Capitolo 4)

Formato:

PCIRCLE *X,Y,raggio[,angolo iniziale,angolo finale]*

Finalità: disegna un cerchio pieno del colore in uso con centro in (X,Y) e con gli altri argomenti specificati.

PEEK (Capitolo 7)

Formato:

PEEK*(indirizzo)*

Finalità: dà il valore che si trova nell'indirizzo di memoria specificato; funzione.

PELLIPSE (Capitolo 4)

Formato:

PELLIPSE *X,Y,raggio orizzontale,raggio verticale[,angolo iniziale,angolo finale]*

Finalità: disegna un'ellisse piena con centro in (X,Y), e con gli altri argomenti.

POKE (Capitolo 4, 6)

Formato:

POKE *cella di memoria,valore dei dati*

Finalità: imposta un valore nella cella di memoria specificata.

POS(0) (Capitolo 3)

Formato:

valore intero = POS(0)

Finalità: dà la posizione orizzontale corrente del cursore di testo; funzione. Lo zero è un argomento "fittizio".

PRINT (Capitolo 2)

Formato:

PRINT *espressione*
? *espressione*

Finalità: visualizza una stringa, un'espressione numerica o una variabile sullo schermo.

PRINT# (Capitolo 7)

Formato:

PRINT *#numero, voce[,voce,...]*

Finalità: invia informazioni al file che è attualmente aperto.

PRINT USING (Capitolo 3)

Formato:

PRINT USING *stringa; variabile o numero[,variabile o numero,...]*

Finalità: stampa una voce sullo schermo usando il formato specificato dalla stringa; potete anche usare PRINT# USING per inviare informazioni al file attualmente aperto, usando un formato predeterminato.

PTSIN (Capitolo 6)

Formato:

PTSIN*(numero)=valore*
POKE PTSIN+*numero*2,valore*

Finalità: variabile usata con le funzioni VDISYS per passare certi argomenti al VDI.

PTSOUT (Capitolo 6)

Formato:

PTSOUT*(numero)*
PEEK(PTSOUT+ *numero*2)*

Finalità: variabile usata per leggere un valore relativo all'output di una operazione VDISYS.

PUT (Capitolo 7)

Formato:

PUT *#numero[,numero del record]*

Finalità: prende le informazioni che sono state memorizzate con LSET o RSET e le mette nel file che specificate. Potete anche specificare il numero del record delle informazioni che state memorizzando.

QUIT (Capitolo 1)

Formato:

QUIT

Finalità: fa in modo che il computer lasci il BASIC ST e ritorni alla scrivania GEM; comando diretto. Assicuratevi di salvare tutto ciò su cui stavate lavorando in BASIC prima di usare questo comando se non volete perdere il materiale.

RANDOMIZE (Capitolo 3)

Formato:

RANDOMIZE *numero*

Finalità: reimposta il generatore di numeri casuali.

READ (Capitolo 2)

Formato:

READ *variabile[,variabile,...]*

Finalità: prende informazioni dalle istruzioni DATA e le memorizza in variabili.

REM (Capitolo 2)

Formato:

REM *commento*

Finalità: fa in modo che il computer ignori il resto della linea di programma, in modo da permettervi d'includere commenti per voi stessi o per altri utenti sul contenuto del programma.

RENUM (Capitolo 2)

Formato:

RENUM *linea nuova,numero della linea iniziale[,incremento]*

Finalità: rinumera il programma in memoria a cominciare dalla linea iniziale, in modo tale che inizi con un nuovo numero e proceda per incrementi che voi specificate; comando diretto.

REPLACE

Formato:

REPLACE *"nome del file"[numero di linea-numero di linea]*

Finalità: salva un programma in memoria, sostituendo il file già esistente su disco; potete specificare quali numeri di linea del programma su disco volete sostituire con quelli attualmente in memoria.

RESTORE (Capitolo 2)

Formato:

RESTORE *[numero di linea]*

Finalità: fà dimenticare al computer dove ha letto l'ultima volta un'istruzione DATA, in modo che quando usa READ per trovare il prossimo dato, comincerà dalla prima istruzione DATA; permette al BASIC di rileggere o riusare interi blocchi d'istruzioni DATA.

RESUME (Capitolo 2)

Formato:

RESUME *[numero di linea]*

Finalità: fa ripartire un programma da una certa linea dopo che è stato riscontrato un errore; questa istruzione funziona associata a ON ERROR GOTO.

RETURN (Capitolo 2)

Formato:

RETURN

Finalità: dopo che è stato emesso un GOSUB, questa istruzione riporta il computer alla linea dov'era quando ha incontrato il GOSUB.

RIGHT$ (Capitolo 3)

Formato:

stringa = RIGHT$*(stringa madre, numero dei caratteri)*

Finalità: funzione; estrae un certo numero di caratteri dalla parte destra di una stringa. La stringa rimane inalterata.

RND (Capitolo 3)

Formato:

variabile numerica = RND*[(numero)]*

Finalità: genera un numero casuale compreso tra 0 e 1; funzione.

RSET (Capitolo 7)

Formato:

RSET *nuova stringa = stringa*

Finalità: funziona come LSET, dato che prepara la stringa per essere memorizzata in un file ad accesso diretto. Tuttavia, se avanzano degli spazi allocati, perché la stringa è più piccola dello spazio allocato, quello che resta della stringa viene riempito di spazi vuoti.

RUN (Capitolo 2)

Formato:

RUN *[numero di linea]*

Finalità: inizia un programma; se specificate un numero di linea, il programma inizierà da quella linea; comando diretto.

SAVE (Capitolo 2)

Formato:

SAVE *"nome del file"*

Finalità: salva il programma che si trova in memoria sul disco con il nome che specificate; comando diretto.

SGN (Capitolo 3)

Formato:

variabile numerica = SGN*(espressione numerica)*

Finalità: dà 1 se il numero è positivo, 0 se il numero è uguale a 0 e —1 se il numero è negativo; funzione.

SIN (Capitolo 3)

Formato:

variabile numerica = SIN*(espressione numerica)*

Finalità: dà il seno di un angolo espresso in radianti; funzione.

SOUND (Capitolo 5)

Formato:

SOUND *voce,volume,nota,ottava,durata*

Finalità: emette una nota musicale o un rumore.

SPACE$ (Capitolo 3)

Formato:

variabile di stringa =SPACE$(*espressione numerica*)

Finalità: stampa il numero specificato di spazi vuoti; funzione.

SPC (Capitolo 3)

Formato:

PRINT SPC(*espressione numerica*)

Finalità: funzione; dà un numero di spazi vuoti in connessione con le istruzioni PRINT, PRINT# o LPRINT.

SQR (Capitolo 3)

Formato:

variabile numerica =SQR(*espressione numerica*)

Finalità: calcola la radice quadrata di un numero; funzione.

STEP (Capitolo 2)

Formato:

STEP

Finalità: mette il BASIC ST nella modalità di depurazione dai bug in modo che ciascuna linea di programma venga eseguita singolarmente e poi si fermi in attesa che premiate il tasto Return; comando diretto.

STOP (Capitolo 2)

Formato:

STOP

Finalità: mette il BASIC ST nella modalità di riposo e ferma il programma.

STR$ (Capitolo 3)

Formato:

variabile di stringa =STR$(*espressione numerica)*

Finalità: funzione; prende un numero e lo converte in stringa.

STRING$ (Capitolo 3)

Formato:

STRING$*(numero,carattere)*

Finalità: funzione; forma una stringa consistente nel numero di caratteri da voi specificato. Il carattere può essere il codice ASCII per quel carattere o il carattere stesso tra virgolette.

SWAP (Capitolo 3)

Formato:

SWAP *variabile1, variabile2*

Finalità: scambia i valori delle due variabili.

SYSTEM (Capitolo 1)

Formato:

SYSTEM

Finalità: fa sì che il computer abbandoni il BASIC ST e che vada alla scrivania GEM; comando diretto.

TAB (Capitolo 3)

Formato:

PRINT TAB *(posizione del tabulato)[“stringa”]*

Finalità: fa spostare il cursore del testo verso destra di un certo numero di spazi; funzione.

TAN (Capitolo 3)

Formato:

variabile numerica =TAN*(espressione numerica)*

Finalità: calcola la tangente di un angolo misurato in radianti; funzione.

TRACE

Formato:

TRACE *numero di linea-numero di linea*

Finalità: visualizza il contenuto delle linee del programma mentre viene eseguito; comando diretto. Potete specificare le linee o i gruppi di linee che volete esaminare. Mentre il programma gira, il BASIC visualizzerà ciascuna linea nella finestra Command mentre viene eseguita. Questo può essere un eccellente mezzo di depurazione dai bug

TROFF

Formato:

TROFF *numero di linea-numero di linea*

Finalità: comando diretto; disattiva il comando TRON. Specificando i numeri di linea, potete anche disattivare specifiche linee del programma.

TRON

Formato:

TRON *numero di linea-numero di linea*

Finalità: fà in modo che il computer visualizzi i numeri delle linee di programma mentre vengono eseguite. Potete specificare i numeri di singole linee o di gruppi di linee che devono essere visualizzati man mano che vengono incontrate. Questo comando diretto, come TRACE, è eccellente per eliminare i bug. Tuttavia, mentre vengono eseguite, non viene visualizzato il contenuto delle linee di programma, ma solo il loro numero.

UNBREAK

Formato:

UNBREAK

Finalità: comando diretto; disattiva il comando BREAK.

UNFOLLOW (Capitolo 2)

Formato:

UNFOLLOW

Finalità: comando diretto; disattiva il comando FOLLOW.

UNTRACE

Formato:

UNTRACE

Finalità: comando diretto; disattivare il comando TRACE.

VAL (Capitolo 3)

Formato:

variabile numerica = VAL*(stringa di cifre)*

Finalità: dà il valore numerico di una stringa; funzione.

VDISYS (Capitolo 6)

Formato:

VDISYS (1)

Finalità: funzione; esegue una funzione dell'interfaccia di dispositivo virtuale (VDI).

WAIT (Capitolo 7)

Formato:

WAIT *porta,numero*

Finalità: fa attendere il computer finché non compare un certo carattere ASCII nella porta che specificate prima che il programma continui.

WAVE (Capitolo 5)

Formato:

WAVE *attivatore[,busta,forma,periodo,ritardo]*

Finalità: altera le caratteristiche del suono che viene prodotto; da solo, WAVE può anche produrre effetti sonori.

WEND (Capitolo 2)

Formato:

WEND

Finalità: chiude un loop WHILE...WEND.

WHILE (Capitolo 2)

Formato:

WHILE *espressione*
  corpo del loop
WEND

Finalità: imposta una condizione in modo che un loop WHILE...WEND continui finché tale espressione non diventi logicamente vera.

WIDTH (Capitolo 7)

Formato:

WIDTH *espressione*

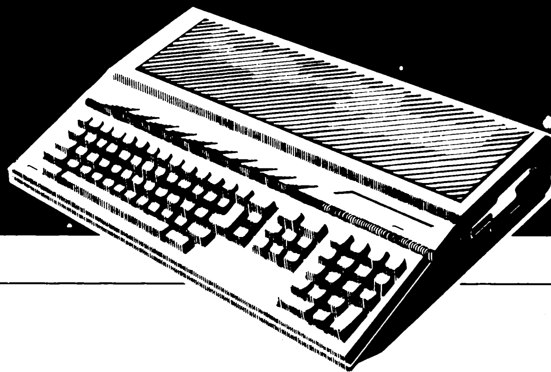
Finalità: cambia la lunghezza della linea sullo schermo; se battete WIDTH LPRINT seguito da un numero, la lunghezza della linea sulla stampante corrisponderà a quel numero di caratteri.

WRITE o WRITE# (Capitolo 7)

Formati:

WRITE *variabile o stringa di dati[,variabile o stringa di dati,...]*
WRITE *#variabile o stringa di dati[,#variabile o stringa di dati,...]*

Finalità: invia informazioni allo schermo o ad un file sequenziale. Le informazioni inviate sono racchiuse tra virgolette e mentre WRITE funziona quasi esattamente come l'istruzione PRINT, WRITE# invia le informazioni al drive.



MESSAGGI D'ERRORE DEL BASIC ST

Quando il BASIC ST incontra un errore di programmazione, visualizza un messaggio d'errore con il numero della linea che lo contiene. Vediamo ora un elenco dei codici di errore che potete incontrare con il BASIC ST, con i relativi significati ed alcune possibili soluzioni per eliminarli.

Error 2

Something is wrong
Qualcosa non va

Sulla maggior parte dei computer questo è conosciuto come errore sintattico, che significa che avete commesso un errore nel formato del comando. Accertatevi che parentesi, virgole, virgolette e tutti gli altri parametri di un comando siano corretti. Purtroppo, il BASIC ST talvolta visualizza questo messaggio così vago anche quando si tratta di errori specificamente definiti.

Error 3

RETURN statement needs a matching GOSUB
Istruzione RETURN richiede un GOSUB corrispondente

Avrete senz'altro almeno un'istruzione RETURN di troppo nel programma. Se il computer incontra un RETURN senza un precedente GOSUB, compare questo messaggio d'errore. Cercate nel vostro programma i GOSUB e i RETURN e fate in modo che il programma eviti il RETURN prima che abbia raggiunto il GOSUB.

Error 4

READ statement ran out of data
Istruzione READ ha esaurito i dati

Non avete messo abbastanza informazioni dopo l'istruzione DATA, avete commesso un errore di dattilografia scrivendo le informazioni (per esempio, avete dimenticato una virgola) oppure state cercando di leggere con READ più di quanto non aveste intenzione di fare. Controllate attentamente tutte le istruzioni READ e DATA ed accertatevi che la quantità d'informazioni che state cercando di leggere sia uguale alla quantità d'informazioni disponibili.

Error 5

Function call not allowed
Chiamata di funzione non permessa

Avete provato una funzione che non esiste o non è permessa.

Error 6

Number is too large
Numero troppo grande

L'ST può trattare numeri solo fino a certe dimensioni, secondo la precisione della variabile numerica.
Quando il numero di byte che avete assegnato definendo il tipo di variabile non è sufficiente per il numero che state creando, incorrerete in questo errore.
Cambiate il tipo di variabile con cui state lavorando (in una a precisione doppia, per esempio) oppure trovate il modo di ridurre le dimensioni del numero.

Error 7

Not enough memory
Memoria insufficiente

Memoria insufficiente a contenere il programma. Riducete il fabbisogno di memoria utilizzando l'istruzione CHAIN oppure snellendo il programma.

Error 8

A statement or a command refers to a nonexistent line
Comando per linea non esistente

Controllate che esistano i numeri di linea referenziati dalle istruzioni GOTO e GOSUB.

Error 9

Subscript refers to element outside the array
L'indice si riferisce ad elemento fuori matrice

State cercando di usare un elemento di una matrice di variabili che non può essere impiegato. Per esempio, se all'inizio del vostro programma ci fosse DIM Y(20) e voi batteste *Y(21)=5*, ricevereste questo messaggio d'errore, perché non avete previsto abbastanza spazio per accogliere questo ventunesimo elemento. Tre possibili soluzioni sono: aumentare lo spazio allocato con DIM, cambiare le variabili che state cercando di usare in modo che rientrino nei limiti delle istruzioni DIM, oppure usare OPTION BASE nel caso che l'elemento 0 abbia causato l'errore.

Error 10

You defined an array more than once
Avete definito una matrice più di una volta

Potete usare DIM solo una volta per riservare lo spazio di una matrice. Se volete cambiare le dimensioni di una matrice, dovrete usare CLEAR per cancellare tutte le variabili oppure ERASE per eliminare la specifica matrice.

Error 11

You cannot divide by zero
Non potete dividere per zero

Controllate il contenuto della linea dove si è verificato questo errore per individuare il punto in cui c'è una divisione con 0 al divisore. Non si può dividere per 0, quindi correggete il programma in modo che il divisore non sia mai uguale a 0.

Error 12

Statement is illegal in direct mode
L'istruzione è illecita nella modalità diretta

Si usa la modalità di comando (o modalità diretta) quando si battono i co-

mandi direttamente, ma le linee del programma vengono conservate in memoria e possono essere esaminate in un secondo tempo. Alcune parole chiave del BASIC ST non possono essere usate direttamente come comandi, ma devono invece essere usate come istruzioni del programma.

Error 13

Types of values do not match
I tipi di valori non sono omogenei

Se tentate un'operazione che prevede un certo tipo di variabile (una stringa, precisione singola, un intero, precisione doppia), ma ne riceve un altro, comparirà questo messaggio d'errore.

Error 15

String cannot be over 255 characters long
Le stringhe non possono essere più lunghe di 255 caratteri

Avete tentato di fare una stringa che supera i 255 caratteri; accorciate la stringa per eliminare il problema.

Error 16

Expression is too long or too complex
L'espressione è troppo lunga o troppo complessa

L'ST non riesce a trattare linee eccessivamente complicate, quindi dovrete dividere la linea in due o tre operazioni più semplici.

Error 17

CONT works only in break mode
CONT funziona solo in modalità di riposo

Lo scopo del comando CONT è di far ripartire un programma dopo che è stato emesso un comando BREAK. Se cercate di usare CONT quando l'ST non è in modalità di riposo, riceverete questo messaggio d'errore.

Error 18

Function needs prior definition with DEF FN
La funzione deve essere definita con DEF FN

Avete cercato di usare una funzione FN senza averla prima definita. Definitela con DEF FN in una parte precedente del programma.

Error 20

RESUME statement found before error routine entered
Istruzione RESUME incontrata prima che fosse inserita una routine d'errore

Se il BASIC non ha incontrato un errore, non c'è alcun motivo di usare RESUME. Eliminate RESUME o fate in modo che il programma lo eviti salvo che si verifichi un errore (con l'istruzione ON ERROR GOTO).

Error 22

Expression has operator with no following operand
L'espressione ha un operatore non seguito da un operando

Avete cercato di emettere un comando, istruzione o funzione con un operatore senza completarlo; per esempio, se battete *PRINT 2+*, il computer si aspetta un altro numero da aggiungere al 2.
Tuttavia, il BASIC ST può segnalarlo semplicemente con "errore sintattico" e far comparire error 2, *Qualcosa non va.*

Error 23

Program line too long
Linea di programma troppo lunga

Invece di avere una linea di programma lunga, dividetela in due o tre linee consecutive.
La lunghezza massima possibile è di 255 caratteri, ma in alcune circostanze potete incontrare questo messaggio d'errore anche per un numero inferiore di caratteri.

Error 30

Window number invalid
Numero di finestra non valido

Il numero usato con OPENW e CLOSEW può solo essere 0, 1, 2 o 3.

Error 31

Argument out of range
Argomento fuori dai limiti

Il numero che state usando è troppo grande; riducetelo.

Error 32

Command cannot be executed from the editor
Il comando non può essere eseguito dall'editor

Avete tentato di mandare un comando mentre eravate in modalità di editing, comando che il computer può usare solo se in modalità di comando o di programma.

Error 33

Line is too complex
Linea troppo complessa

La formula che state usando nella linea di programma è troppo complessa; spezzatela in due o tre linee distinte.

Error 50

FIELD statement caused overflow
Istruzione FIELD causa un'eccedenza

Se cercate di allocare troppi byte per una variabile, incorrerete in questo errore.

Error 51

Device number invalid
Numero di dispositivo non valido

Avete cercato di usare l'istruzione OPEN con un numero di dispositivo che non esiste.

Error 52

File number or filename invalid
Numero o nome di file non valido

Avete cercato di far riferimento ad un file con un nome o un numero che o non esiste o non è aperto.

Error 53

File not found on disk drive specified
File non trovato sul drive specificato

Avete cercato di caricare un file che non si trova sul disco. Usate il comando DIR per vedere il contenuto del disco attualmente nel drive.

Error 54

File mode not valid
Tipo di file non valido

I file sequenziali e i file ad accesso diretto non possono esser mescolati e se usate PUT, GET, LOF o altri comandi di file con il tipo sbagliato di file, incorrerete in questo errore.

Error 55

You cannot OPEN or KILL a file already open
Non potete usare OPEN o KILL con un file già aperto

Il file attualmente aperto deve essere chiuso prima che possiate riaprirlo o eliminarlo.

Error 57

Disk input/output error
Errore input/output di disco

Può darsi che il vostro disco sia difettoso, che il drive stia funzionando male o forse il cavo del drive non è ben fissato. Controllate l'hardware quando si verifica questo errore e se il sistema sembra in ordine, dovrete resettare il computer per reiniziare.

Error 58

File exists
Il file esiste

Avete cercato di salvare su disco un file che già esiste. Usate un altro comando (ad esempio REPLACE) per rimpiazzare il file su disco con quello che avete in memoria.

Error 61

Disk is full
Il disco è pieno

Non c'è abbastanza spazio sul disco per salvare il file. Usate un altro disco o eliminate i file che non servono dal disco in uso.

Error 62

You have reached end-of-file
Avete raggiunto la fine del file

Se state caricando i record di un file sequenziale, riceverete questo messaggio d'errore se cercherete di superare la fine del file. Usate la funzione EOF per evitare questo errore.

Error 63

The record number in PUT or GET is more than 32767 or zero
Il numero del record in PUT o GET è maggiore di 32767 o è zero

Non potete specificare un numero di record minore di 1 e maggiore di 32767.

Error 64

Invalid filename
Nome di file non valido

Questo di solito è causato dall'avere troppi caratteri nel nome del file. Date un nome diverso al file che state cercando di caricare, salvare o cancellare.

Error 65

Invalid character (character number) in program file
Carattere (o numero di carattere) non valido

Se cercate di caricare un file che contiene dati difettosi (magari a causa di un disco scadente), comparirà questo messaggio d'errore. Provate a risalvare il file e verificatene l'accuratezza oppure modificate i dati del file in modo da eliminare tutti i caratteri non validi. Ricordate che i dati in un file devono essere ordinati in modo che corrispondano a ciò che deve essere letto; non potete leggere una stringa in una variabile numerica, per esempio.

Error 99

-Break-

Quando entrate in modalità di riposo (a seguito di un comando BREAK o STOP), verrà visualizzato questo messaggio d'"errore". Non indica che c'è qualcosa di sbagliato. Battete *CONT* per far ripartire il programma.

Error 101

Program has too many lines
Il programma ha troppe linee

Avete esaurito la memoria; eliminate alcune linee non necessarie o spezzate il programma in diverse parti e collegatele con un'istruzione CHAIN.

Error 102

Statement is out of range
L'istruzione eccede i limiti

La variabile usata con l'istruzione ON è troppo grande per poter essere associata ai numeri di linea da voi specificati. Se A è uguale a 5, per esempio, e battete *ON A GOTO 100,200*, il computer non saprà dove andare, poiché A non era uguale a 1 o a 2. Scoprite perché la variabile è più grande di quanto volete che sia.

Error 103

Invalid line number
Numero di linea non valido

Avete cercato di usare un numero di linea superiore a 65529, che è il limite massimo.

Error 104

A variable is required
È richiesta una variabile

Avete cercato di usare una funzione matematica senza includere la necessaria variabile. Controllate la linea e aggiungete la variabile che volete usare con la funzione matematica.

Error 106

Line number does not exist
Il numero di linea non esiste

Il programma fa riferimento ad un numero di linea che non esiste con GOTO, GOSUB o un'altra istruzione.

Error 107

Number too large for an integer
Numero troppo grande per un intero

Gli interi possono essere compresi tra —32768 e 32767; se cercate di uscire da questi limiti con una variabile intera, incorrerete nell'errore 107.

Error 108

Input data is not valid, restart input from first item
I dati input non sono validi, reiniziate l'input dalla prima voce.

Avete cercato d'inserire dati che non corrispondono con quello che l'istruzione INPUT richiede. Se INPUT attende una variabile numerica, non accetterà una stringa al suo posto, e INPUT non accetterà nemmeno dati in eccesso o in difetto quando ne richiede un certo numero.

Error 109

Stop

Come Error 99, questo non è un errore vero e proprio; indica semplicemente che ha incontrato un'istruzione STOP.

Error 110

You have nested subroutine calls too deep
Avete annidato chiamate di subroutine troppo in profondità

Avete probabilmente fatto troppe chiamate GOSUB senza RETURN. Eliminate alcuni GOSUB e trovate un modo diverso di ottenere lo stesso risultato.

Error 111

Invalid BLOAD file
File BLOAD non valido

BLOAD carica un file binario e se cercate di caricare un tipo diverso di file con questo comando, si verifica questo errore.

Error 202

Command not allowed here
Comando non possibile qui

Alcuni comandi non funzionano in certe modalità; ERA, per esempio, non funziona all'interno di un programma. Eliminate il comando e trovatene un altro con cui sostituirlo, oppure passate ad un'altra modalità.

Error 203

Line number is required
Serve il numero di linea

Se cercate di usare una parola chiave permessa solo in un programma nella modalità di comando, si verifica questo errore.

Error 204

FOR statement needs a NEXT or WHILE needs a WEND
Un'istruzione FOR richiede un NEXT o WHILE richiede WEND

Non ci sono abbastanza istruzioni NEXT o WEND in corrispondenza d'istruzioni FOR e WHILE.

Error 205

NEXT statement needs a FOR or WEND needs a WHILE
Un'istruzione NEXT richiede un FOR o WEND richiede WHILE

Avete troppe istruzioni NEXT o WEND rispetto alle istruzioni FOR e WHILE.

Error 206

A comma is expected
È necessaria una virgola

Se l'istruzione INPUT riceve dati non separati dalle virgole, verrà visualizzato questo messaggio d'errore.

Error 207

A parenthesis is expected
È necessaria una parentesi

L'istruzione DEF FN richiede le parentesi, e voi ne avete dimenticata una.

Error 208

OPTION BASE must be 0 or 1
OPTION BASE deve essere 0 o 1

L'elemento più basso in una matrice di variabili può essere 0 o 1; voi avete cercato d'impostare l'elemento più basso ad un numero diverso da 0 o 1 con OPTION BASE.

Error 209

Too many arguments in your list
Troppi argomenti nella vostra lista

Avete incluso più argomenti di quanti siano necessari. Eliminate gli argomenti superflui.

Error 213

Function defined more than once
Funzione definita più di una volta

Se cercate di usare DEF FN con lo stesso nome di funzione più di una volta,

viene visualizzato questo messaggio d'errore. Cambiate il nome della seconda definizione di funzione o eliminatela.

Error 214

You are trying to jump into a loop
State cercando di entrare in un loop

Non potete entrare in una parte di un programma compresa tra FOR e NEXT oppure tra WHILE e WEND.

Error 221

System error # number, please restart
Errore di sistema # numero, per favore ricomincia

Si è verificato un errore piuttosto grave e dovete riavviare il BASIC ST dalla scrivania GEM.

Error 222

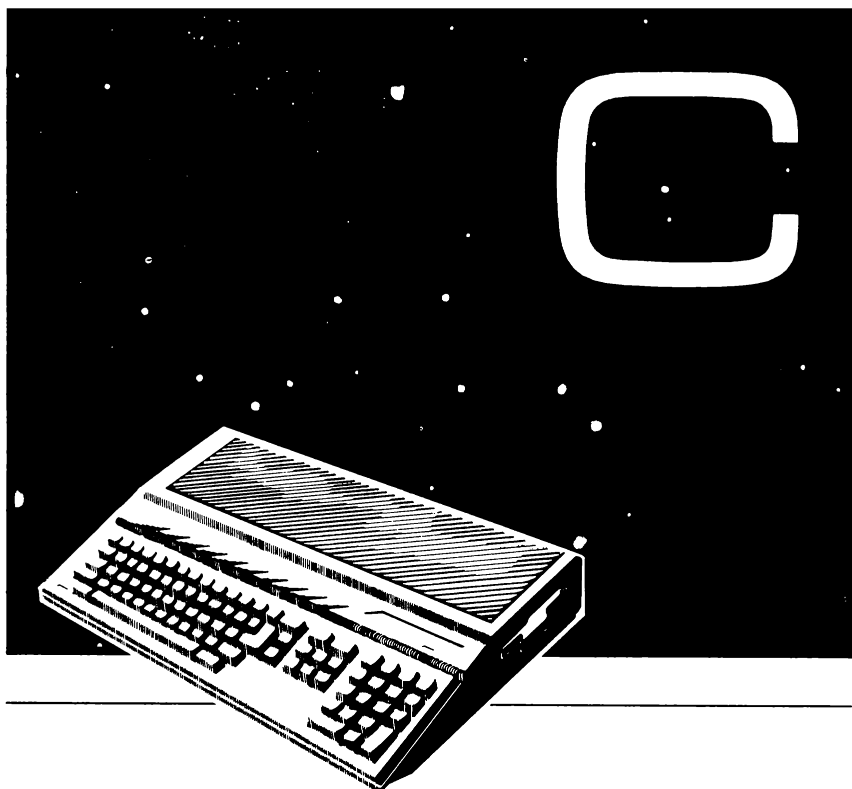
Program not run
Programma non lanciato

Se il computer scopre un errore prima che inizi a lanciare il programma, comparirà questo messaggio in modo che possiate passare nella modalità di editing e trovare l'errore.

Error 223

Too many FOR loops
Troppi loop FOR

State annidando troppi loop FOR; eliminatene alcuni.



INSTALLAZIONE

Quello che segue è un breve profilo dei passaggi necessari per mettere in opera un computer ST.

MESSA IN OPERA

1. Togliete il monitor, la tastiera e (se avete un 520ST) il drive dall'imballaggio e poneteli su una superficie piatta e stabile.
2. Assicuratevi che tutti gli interruttori siano in posizione off, poi collegate il cavo principale di alimentazione dalla console della tastiera (contrassegnato da *Power*) ad una presa.
3. Collegate il mouse alla porta 0 sul lato destro dell'ST.
4. Mettete il monitor nella posizione che gli avete assegnato, quindi collegate il cavo del video dal monitor alla console della tastiera. Collegate poi il cavo di alimentazione dal monitor ad una presa.
5. Se avete un 520ST o un drive esterno extra, collegate il cavo dell'interfaccia del drive dalla console della tastiera al drive. Inoltre collegate il cavo di alimentazione del drive dalla porta di alimentazione del primo drive ad una presa.
6. Se avete due drive esterni, collegate il cavo di aimentazione del secondo drive dal drive ad una presa, poi usate il cavo dell'interfaccia per collegare il secondo drive al primo.

COME PROTEGGERE IL VOSTRO SISTEMA

A questo punto, il vostro sistema dovrebbe essere completamente collegato. Prima di procedere, assicuratevi che ogni componente sia collocato su una superficie piana e stabile, in modo che non ci sia il rischio che qualcosa possa cadere. La collocazione del sistema è importante, quindi tenete presenti i seguenti punti.

- Fate in modo di non inciampare nei cavi delle interfacce e di alimentazione. Se ciò accade il computer potrebbe spegnersi cancellando tutte le informazioni in memoria e senza contare che qualche parte del computer potrebbe cadere in terra.
- Tenete bevande, cibi, fumo e bambini piccoli alla larga dal sistema. L'ST

è uno strumento importante, quindi vale la pena di tribolare un pò per evitare di dover sprecare tempo (e denaro) in riparazioni.

- Se l'elettricità in casa vostra è un pò imprevedibile, è forse meglio che prendiate delle precauzioni. Con una spesa dell'ordine delle 50mila lire, potete acquistare un dispositivo da innestare nella presa che usate per il computer per proteggerlo da sbalzi di corrente. Ci sono anche dispositivi che servono da temporanea fonte di alimentazione in caso vi sia una sospensione della corrente, dandovi abbastanza tempo per salvare le informazioni importanti. Tuttavia, resta più economico salvare spesso i file su disco e farne una copia di riserva nell'eventualità che l'originale subisca dei danni. Non c'è niente di peggio che perdere delle informazioni al computer, dato poi che non c'è praticamente alcun modo di recuperarle. I vantaggi che il computer offre sono pari agli svantaggi che potrete avere se non lo usate nei modi dovuti.

COME PARTIRE

Accendete il computer; prima il monitor, poi il drive e infine l'unità della tastiera. Se avete un vecchio 520ST senza il chip di ROM, il monitor visualizzerà il messaggio che vi invita a mettere il System Disk nel drive. Eseguite e l'ST lo caricherà in memoria. In futuro potrete mettere il disco nel drive ancor prima di accendere il computer, in modo da evitare la pausa del messaggio. Quando il System Disk è stato completamente caricato, compariranno sullo schermo quattro figure: le icone del drive A, del drive B, del cestino e la barra dei menù. Vi trovate nella Scrivania GEM.

Se avete un 1040ST o una versione nuova del 520ST (quelle vendute a partire dal febbraio 1986), basta accendere prima il monitor, poi il drive esterno (se c'è) ed infine il computer stesso. Comparirà la scrivania GEM. La scrivania comparirà più in fretta se avete già un disco, il Language Disk, o un disco applicativo o di un vostro programma, inserito nel drive prima di accendere il sistema. In ogni caso, dovrete comunque inserire un disco una volta che compare la scrivania se volete andare avanti.



I CODICI DEI CARATTERI ASCII

La tabella seguente elenca tutti i caratteri ed i rispettivi numeri nella versione Atari ST dell'American Standard Code for Information Interchange, ovvero Codice Standard Americano per l'Interscambio di Informazioni. Come abbiamo già detto nel Capitolo 3, potete usare la funzione ASC per convertire un carattere nel suo valore numerico ASCII e la funzione CHR$ per convertire un numero in un carattere.

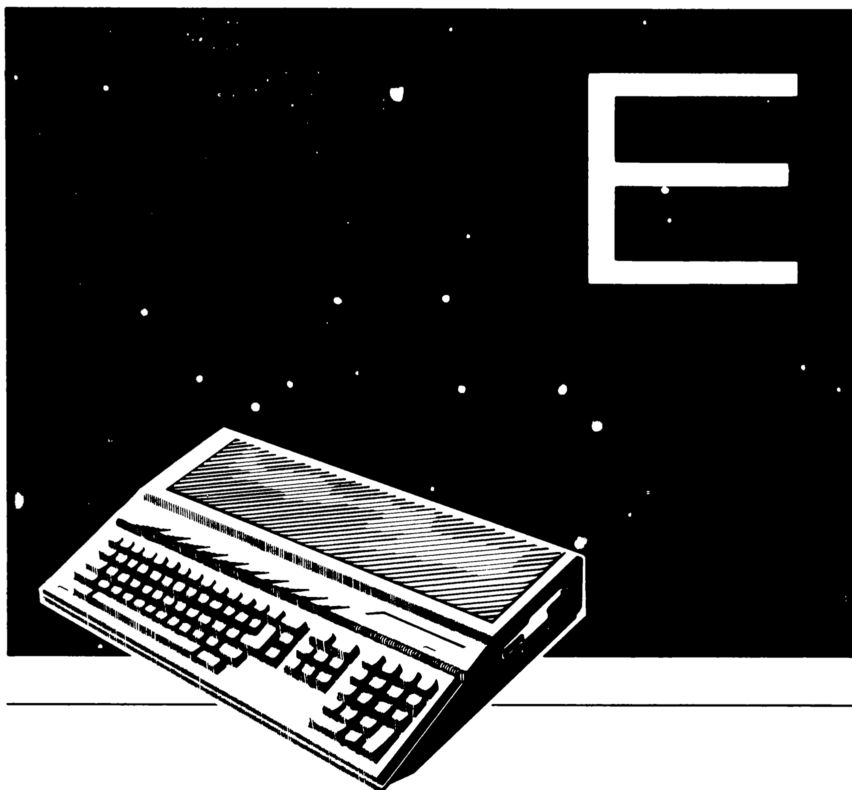
Ai seguenti numeri non corrispondono caratteri stampati:

0	Niente
7	Campanello
10	Avanzamento linea
13	Ritorno carrello
32	Spazio

Valore ASCII	Carattere	Valore ASCII	Carattere
		35	#
1	⇧	36	$
2	⇩	37	%
3	⇨	38	&
4	⇦	39	'
5	☒	40	(
6	◩	41	)
7		42	*
8	✓	43	+
9	◷	44	,
10		45	-
11	♪	46	.
12	[illegible]	47	/
13		48	0
14	J|	49	1
15	|L	50	2
16	0	51	3
17	1	52	4
18	2	53	5
19	3	54	6
20	4	55	7
21	5	56	8
22	6	57	9
23	7	58	:
24	8	59	;
25	9	60	<
26	ə	61	=
27	ɛ	62	>
28	[illegible]	63	?
29	[illegible]	64	@
30	[illegible]	65	A
31	[illegible]	66	B
32		67	C
33	!	68	D
34	"	69	E

Valore ASCII	Carattere	Valore ASCII	Carattere
70	F	126	~
71	G	127	∆
72	H	128	ç
73	I	129	ü
74	J	130	é
75	K	131	â
76	L	132	ä
77	M	133	à
78	N	134	å
79	O	135	ç
80	P	136	ê
81	Q	137	ë
82	R	138	è
83	S	139	ï
84	T	140	î
85	U	141	ì
86	V	142	Ä
87	W	143	Å
88	X	144	É
89	Y	145	æ
90	Z	146	Æ
91	[	147	ô
92	\	148	ö
93	]	149	ò
94	^	150	û
95	_	151	ù
96	`	152	ÿ
97	a	153	Ö
98	b	154	Ü
99	c	155	¢
100	d	156	£
101	e	157	¥
102	f	158	ß
103	g	159	ƒ
104	h	160	á
105	i	161	í
106	j	162	ó
107	k	163	ú
108	l	164	ñ
109	m	165	Ñ
110	n	166	ª
111	o	167	º
112	p	168	¿
113	q	169	⌐
114	r	170	¬
115	s	171	½
116	t	172	¼
117	u	173	¡
118	v	174	«
119	w	175	»
120	x	176	ã
121	y	177	õ
122	z	178	Ø
123	{	179	ø
124	|	180	œ
125	}	181	Œ

Valore ASCII	Carattere	Valore ASCII	Carattere
182	À	219	ף
183	Ã	220	ץ
184	ð	221	§
185	·	222	^
186	´	223	∞
187	'	224	α
188	¶	225	β
189	©	226	Γ
190	®	227	π
191	™	228	Σ
192	ĳ	229	σ
193	Ĳ	230	µ
194	א	231	τ
195	ב	232	Φ
196	ג	233	Θ
197	ד	234	Ω
198	ה	235	δ
199	ו	236	∅
200	ז	237	φ
201	ח	238	∈
202	ט	239	∩
203	י	240	≡
204	כ	241	±
205	ל	242	≥
206	מ	243	≤
207	נ	244	⌠
208	ס	245	⌡
209	ע	246	÷
210	פ	247	≈
211	צ	248	°
212	ק	249	•
213	ר	250	·
214	ש	251	√
215	ת	252	ⁿ
216	ן	253	²
217	ך	254	³
218	ם	255	¯



NUOVE PAROLE CHIAVE DEL BASIC ST

Nel momento in cui questo libro stava andando in stampa, la Atari progettava di aggiungere alcune nuove parole chiave al BASIC ST. Era previsto che questa nuova versione del linguaggio venisse inclusa nel Language Disk che accompagna i nuovi computer ST in vendita a partire da Settembre 1986. Il responsabile delle pubblicazioni all'Atari è stato così gentile da fornirmi una descrizione di queste nuove istruzioni del BASIC ST. Poiché la Atari ha ricevuto la versione finale del software dai suoi programmatori in Inghilterra solo pochi giorni prima che questo libro andasse in stampa, questo materiale viene presentato in un'Appendice. Inoltre, dato che io non disponevo del software da provare, né io né l'editore possiamo garantire che le nuove parole chiave funzionino esattamente come descritto. Tuttavia le descrizioni seguenti dovrebbero rivelarsi comunque utili per coloro che hanno la nuova versione del BASIC ST.

AREA

Formato:

AREA *X1,Y1,X2,Y2,X3,Y3[.Xn,Yn]*

Finalità: comando diretto; disegna sullo schermo un poligono usando i punti da voi specificati e lo riempie con il colore in uso. Dato che un poligono è formato da tre o più punti, dovete specificare almeno tre coppie (X,Y). Per esempio per disegnare un poligono con i vertici in (20,25), (35,43), (100,125), (500,6) e (200,76), dovrete battere:

AREA 20,25,35,43,100,125,500,6,200,76

ASK MOUSE

Formato:

ASK MOUSE *X,Y,B*

Finalità: funzione; vi dà le coordinate X e Y del puntatore e la posizione del bottone del mouse. Usate ASK MOUSE seguito da tre variabili. Non è indispensabile che le chiamiate X, Y e B, ma sarà più facile ricordarne il significato se le chiamate così.

ASK RGB

Formato:

ASK RGB *colore,r,g,b*

Finalità: funzione; conserva i livelli di rosso, verde e blu di un colore della tavolozza nelle variabili da voi specificate. Per esempio, per trovare i livelli di colore del colore numero 3 e conservarli nelle variabili A, B e C, basta battere:

ASK RGB 3,A,B,C

BOX

Formato:

BOX *[FILL] X1,Y1;X2,Y2*

Finalità: comando; disegna sullo schermo un rettangolo con l'angolo in alto a sinistra nel punto (X1,Y1) e l'angolo in basso a destra in (X2,Y2). Se inserite FILL immediatamente dopo BOX, il rettangolo verrà riempito del colore in uso. I seguenti comandi disegnano un rettangolo vuoto e un rettangolo pieno aventi le stesse coordinate:

BOX 10,20;50,67
BOX FILL 10,20;50,67

DRAW

Formato:

DRAW *X1,Y1,X2,Y2[.Xn,Yn]*

Finalità: comando diretto; disegna sullo schermo una serie di linee usando i punti specificati. Per usare questo comando, battete *DRAW* seguito da tutte le coppie di coordinate che volete usare. DRAW funziona come un disegno per punti; il ST parte dal primo punto e passa ai successivi nell'ordine in cui li avete specificati. Per esempio, per disegnare una linea da (10,15) a (76,57) e poi da (76,57) a (45,30), dovrete battere:

DRAW 10,15,76,57,45,30

DRAWMODE

Formato:

DRAWMODE *numero della modalità*

Finalità: istruzione; se mettete un'immagine sullo schermo usando DRAW, DRAWMODE dirà al computer come trattare quell'oggetto in relazione al resto della grafica sullo schermo. Le quattro modalità di disegno sono:

Numero della modalità	Finalità
1	Sostituisce (l'oggetto che disegnate copre qualsiasi immagine ci sia già)
2	Trasparente (qualsiasi immagine precedente rispetto al vostro oggetto resta visibile)
3	OR esclusivo
4	Reverse trasparente (l'oggetto che disegnate compare dietro alle immagini già sullo schermo)

GSHAPE

Formato:

GSHAPE *X,Y,matrice*

Finalità: comando diretto; prende l'immagine grafica che avete conservato in memoria (chiamata *raster*) e la porta nella posizione dello schermo da voi specificata. Il raster è un'area grafica rettangolare e le coordinate (X,Y) che specificate dopo GSHAPE dicono al computer dove deve collocare l'angolo in alto a sinistra del raster. L'argomento *matrice* è semplicemente la matrice che contiene il raster. Questo comando opera insieme a SSHAPE, che salva il raster dallo schermo in memoria.

LINEPAT

Formato:

LINEPAT *stile[,modello]*

Finalità: funzione; cambia il modello usato per le linee che tracciate sullo schermo. Ci sono sette stili diversi che potete specificare dopo LINEPAT:

Numero	Stile
1	Pieno
2	Lineette lunghe
3	Punto
4	Lineetta punto
5	Lineetta
6	Lineetta punto punto
7	Modello definito dall'utente

Se usate lo stile 7, dovete inserire un secondo numero corrispondente al modello che volete. Per inserire un vostro modello, determinatene il numero a 16 bit (con gli 1 in corrispondenza dei bit grafici accesi e gli 0 in corrispondenza dei bit spenti) e traducetelo in decimale. Un pattern come 0101010101010101 si traduce in 13,653, e battendo:

```
LINEPAT 7,13653
```

metterete questo modello di linea in memoria.

MAT AREA

Formato:

MAT AREA *numero,matrice*

Finalità: comando diretto; simile ad AREA, salvo che usa diversi vertici (specificati dal primo argomento) registrati in una matrice (specificata dal secondo argomento). Se aveste 12 coppie di coordinate conservate in H%(), quindi, si dovrebbe battere *MAT AREA 12,H%()* per disegnare sullo schermo un'area impiegando quei punti. Come con AREA, il poligono è pieno del colore in uso. Potete specificare fino a 128 diverse coppie di coordinate quando usate MAT AREA.

MAT DRAW

Formato:

MAT DRAW *numero,matrice*

Finalità: comando diretto; simile a MAT AREA, tranne che il poligono non è pieno. Potete anche sostituire MAT LINEF a questo comando.

PATTERN

Formato:

PATTERN *piano,matrice*

Finalità: funzione; crea una colorazione di fondo definita dall'utente. Battete *PATTERN* seguito dal numero del *piano* che corrisponde al numero degli elementi della matrice che avete usato:

Numero di piano	Numero di elementi
1	16
2	32
4	64

Poi battete il nome della *matrice* i cui elementi definiscono il pattern. Per esempio, supponiamo che abbiate messo 32 numeri nella matrice Y%(), per definire il motivo di riempimento che volevate. Per caricarlo in memoria dovrete battere *FILL 2,Y%()*.

Come con LINEPAT, si determinano i numeri per il motivo che volete scrivendo i numeri binari che corrispondono ai pixel che volete accesi o spenti. Poi si traducono questi numeri binari in decimali e si collocano in una matrice. I numeri binari dovrebbero essere lunghi 16 bit.

PI

Formato:

variabile# =PI

Finalità: funzione; colloca il valore pi in una variabile. Battete il nome della variabile (che dovrebbe essere definita a precisione doppia facendola seguire dal segno #) seguito da *=PI.* Potete anche usare pi da solo in una formula matematica. Pi è approssimativamente uguale a 3.1415926538979.

RGB

Formato:

RGB *numero del colore,rosso,verde,blu*

Finalità: istruzione; imposta i livelli di rosso, verde e blu di un colore della tavolozza. Battete RGB seguito dal numero del colore che volete cambiare e poi aggiungete i livelli di rosso, verde e blu (ciascuno dei quali può variare da 0 a 7). Per esempio, per cambiare il colore numero 5 in puro rosso, batterete:

```
RGB 5,7,0,0
```

SSHAPE

Formato:

SSHAPE *X1,Y1,X2,Y2,matrice*

Finalità: comando diretto; registra un raster (un'area grafica rettangolare) in memoria. Battete *SSHAPE* seguito dalle coordinate dell'angolo in alto a sinistra del raster *(X1,Y1)*, dell'angolo in basso a destra e della *matrice* nella quale volete conservare il raster. Per memorizzare la regione rettangolare definita dagli angoli (25,50) e (367,132) nella matrice S%() dovrete inserire:

```
SSHAPE 25,50,367,132,S%()
```

Ricordate che la matrice deve essere dimensionata (usando l'istruzione DIM) in modo da poter allocare abbastanza spazio per il raster. Dato che di solito è meglio riservare uno spazio abbastanza grande (per esempio 500 o 1000 elementi), accertatevi che il programma non esaurisca la memoria per registrare il raster.

INDICE ANALITICO

COLLANA INFORMATICA

Autore/Titolo	ISBN 88 7081
Atkinson L.V. - Pascal. Corso di programmazione per microcalcolatori	132 8
Banahan M., Rutter A. - Unix. Introduzione al sistema operativo per microcalcolatori	144 1
Benasi A.L. - Introduzione ai personal computer	161 1
Berk A.A. - LISP. Il linguaggio dell'intelligenza artificiale	240 5
Bowmann D.J. - Introduzione al CAD/CAM	212 X
Brooner E.G., Wells P. - Computer. Sistemi di comunicazione	220 0
Buffington C. - Il tuo primo personal computer. Come sceglierlo e usarlo	154 9
Burkinshaw C.I., Goodley R. - MSX Linguaggio macchina	252 9
Checroun A. - Basic. Corso di programmazione per microcalcolatori	149 2
Czerwinski M. - Hardware. 83 test	169 7
Czerwinski M. - Software. 96 test	170 0
Daris F.E., Barry J., Wiesenberg M. - Editoria da ufficio coi personal computer	322 3
Diemer W.D. - Primo corso di CAD	321 5
Dreyfus M. - Fortran IV	168 9
Eigner M., Maier H - CAD (Computer Aided Design). Impiego dei sistemi di progettazione assistita da calcolatore	123 9
Gallippi A. - Glossario di informatica	258 8
Gallippi A. - Introduzione all'informatica - 3ª ed.	216 2
Gallippi A. - Piccoli calcolatori	231 6
Honerkamp M., Jetter M. - Il simulatore di volo per Apple //, PC IBM e Commodore 64	291 X
Kaier E. - MSX Linguaggio e programmazione	234 0
Kerler C. - Donne e computer	245 6
Kipnis G. - Statistica in BASIC	210 3
Knight T. - Atari ST - Programmazione in BASIC	325 8
Laurent J.P. - Analisi e programmazione strutturata. Concetti fondamentali - 2ª ed.	364 9
Laurent J.P., Ayel J. - Analisi e programmazione strutturata. Esercizi commentati	308 8
Macchi C., Guilbert J.F. - Telematica. Trasporto e trattamento dell'informazione	084 4
Martin D. - Banca dati. Applicazioni sui piccoli e grandi calcolatori	087 9
Matuszek D.L. - Pascal veloce	152 2
Mayoh B. - Programmare con il linguaggio Ada	120 4
Naylor C. - Programmazione semplice dei generatori di programmi	200 6
Naylor C. - Sistemi esperti per il vostro personal computer	191 3
Norris D.E., Skilbeck C.E., Hayward A.E., Torpy D.M. - Microcomputer e applicazioni ospedaliere	243X
Pariot C. - Introduzione ai microprocessori e ai microelaboratori	044 5
Pujolle G. - Telematica, reti e applicazioni	237 5
Realini G. - Disegnare col computer - 2ª ed.	218 9
Rivière J. - Programmare in Assembler	124 7
Rogers D.F. - Principi di programmazione grafica	307 X
Sartori L.G. - Informatica in fabbrica	184 0
Scanlon L.J. - Programmazione del microprocessore 68000	219 7
Schwinn R. - dBase II	242 1
Simons G.L. - Intelligenza artificiale, la nuova frontiera dell'informatica	201 4

Simons G.L. - Verso la quinta generazione	232 4
St John Bate J., Burgess R. - Informatica in ufficio (Office automation)	229 4
Thoma J. - Progettare con i bondgraph	064 X
Townsend C. - dBase III. Programmazione e applicazioni	225 1
Voisinet D.D. - CAD	282 0
Vuldy J.L. - Grafica tridimensionale per il personal computer	186 7
Waite M., Martin D., Prata S. - Unix System V. Sistemi operativi	214 6
Waite M., Prata S., D. Martin - Programmare in C	215 4
Winfield A. - Forth. Corso di programmazione per microcalcolatori	140 9
Wirth N. - Algoritmi + Strutture dati = Programmi	259 6
Grafica col computer	*185 9*
Programmi per computer MSX	*224 3*

IBM PC E COMPATIBILI

Baras E.M. - Manuale d'uso Lotus 1-2-3 (versione italiana 2.0)	326 6
Beil D.H. - IBM PC. Visicalc	163 8
Brown J.R., Finkel L.R. - IBM PC. Gestione degli archivi di dati	176 X
Brown J.R., Finkel L.R. - IBM PC. Gestione degli archivi di dati (Programmi registrati su disco)	812 8
Coffron J.W. - IBM PC. Connessioni e interfacce	202 2
Conklin D. - IBM PC. Grafica	146 8
Conklin D. - IBM PC. Grafica (Programmi registrati su disco)	814 4
Cowart - dBASE III Plus	337 1
Fernandez J.N., Ashley R. - IBM PC. Impiego del CP/M-86	145 X
Hoffman P. - Microsoft Word versione per MS-DOS	255 3
Kolodney D., Blackadar T. - WordStar 2000	266 9
Lasselle J., Ramsay C. - Introduzione al personal computer IBM	157 3
Manchon F., Nasr J.M. - Fisica su personal computer (+ disco IBM-M24)	334 7
Poole L. - IBM PC. Guida all'impiego - 2ª ed.	267 7
Poole L. - 35 programmi in BASIC per l'IBM PC	165 4
Press L. - IBM PC. Applicazioni professionali	190 5
Reverchon A., Ducamp M. - Matematica su personal computer. 1 Analisi (+ disco IBM-M24)	287 1
Reverchon A., Ducamp M. - Matematica su personal computer. 2 Algebra (+ disco IBM-M24)	288 X
Senftleben D. - Logo per IBM PC e M24	236 7
Simpson A. - Framework. Software integrato per IBM PC e compatibili	250 2
Simpson A. - IBM PC. Gestione dei file	180 8
Simrin S. - Sistema operativo MS-DOS	257 X
Willen D.C., Krantz J.I. - IBM PC. Programmazione del microprocessore 8088	221 9
Williams A.T. - IBM PC. Fogli elettronici	177 8
Withe R.B. - Wordstar (versione italiana 3.4)	162 X
IBM PC e compatibili. 18 programmi (listati + disco)	209 X

APPLE

Albanesi R. - Jazz per Macintosh	253 7
Bitter G.G., Watson N.R. - Il Logo per l'Apple //	155 7
Hoffman P. - Microsoft Word versione per Macintosh	256 1
Kascmer J. - Una guida facile all'Apple //	156 5
McComb G. - Macintosh. Guida all'uso	207 3
Poole L. - 35 programmi in BASIC per l'Apple //	164 6
Poole L., Borchers M., Castlewitz D.M. - Apple II. 72 programmi	151 4
Samish F. - Macintosh. Come e dove usarlo	198 0

COMMODORE

Andrews M. - Commodore 64/128. Programmazione in Assembly	304 5
Finkel S. - Commodore 16. Manuale d'uso	205 7
Harrison M. - Come gestire meglio il Commodore 64	159 X
Highmore D., Page L. - Programmazione semplice del Commodore 64	160 3
Meyer S.C. - Commodore Plus/4. Guida all'uso	213 8
Rugheimer H., Spanik C. - Commodore 64. Manuale PEEK-POKE	228 6
Sinclair I. - Commodore 64. Codice macchina	178 6
Sinclair I. - Commodore 64. Guida all'uso	166 2
Stewart J. - Commodore 64 per la scuola e il tempo libero. Libro 1	226 X
Stewart J. - Commodore 64 per la scuola e il tempo libero. Libro 2	227 8
Waite M., Lafore R., Volpe J. - Commodore 128 personal computer	251 0
Musica col Commodore 64	230 8

SOFTWARE PROFESSIONALE

Bazzocchi G. - Oleocalc. Progettazione di impianti oleodinamici (PC IBM e compatibili)	822 5
Castellani G. - Programma IPAR/Programmi IRID-RH-RF (Olivetti M21-M24 e compatibili)	823 3
Norris D.E., Skilbeck C.E., Hayward A.E., Torpy D.M. - Microcomputer e applicazioni ospedaliere (Apple //)	830 6
F. Calza, L. Dozio, D. Mazzanti - Prog. 373 (PC IBM e compatibili)	827 6
G. Castellani - Calcolo dei cuscinetti e momenti flettenti (Olivetti M24 e compatibili)	835 7
C. Confalonieri - Tubisoft (PC IBM e compatibili)	836 5
Screen case (PC IBM e compatibili)	838 1

DELLO STESSO EDITORE

COLLANA DI ELETTRONICA ED ELETTROTECNICA

223 5	Fibre ottiche - W.T. Boyd
222 7	Le comunicazioni televisive via satellite - C. Bowick, J. Kearney
274 X	Circuiti logici TTL - D. Lancaster
275 8	Circuiti logici CMOS - D. Lancaster
264 2	Applicazioni industriali dell'Elettronica - J.L. Austin, B. Barrier
276 6	Amplificatori operazionali integrati - R. Melen, H. Garland
277 4	Circuiti digitali - R.G. Middleton
119 0	Manuale di Elettronica - D.G. Fink, D. Christiansen
289 8	Lettura e interpretazione degli schemi - D. Herrington
270 7	Elettronica 1 - Elementi di Elettrotecnica - H. Meister
271 5	Elettronica 2 - Componenti e Dispositivi - K. Beuth
268 5	Elettronica 3 - Circuiti e Sistemi analogici (+disco) - K. Beuth, W. Schmusch
269 3	Elettronica 4 - Circuiti e Sistemi digitali (+disco) - K. Beuth
301 0	Circuiti fondamentali a transistori - C.A. Pike
319 3	I Microprocessori - D.L. Cannon, G. Luecke
302 9	Sistemi per l'automazione - N.M. Schmitt, G. Luecke
320 7	Sistemi flessibili di produzione - H.J. Warnecke, R. Steinhilper
083 6	Applicazioni industriali dei laser - J.F. Ready
073 9	Manuale motori elettrici - E.H. Werninck
077 1	Azionamenti elettrici - W. Boehm
265 0	Manuale degli impianti elettrici - H.G. Boy, V. Dunkhase


CHIP
Mensile di micro e personal computer
Marzo 1985 N. 3 L. 3.000
Lotus Symphony
Commodore Plus/4
Spectravideo SVI 728
Compaq Deskpro
Grafica computerizzata
Hollywood all'elaboratore

Una completa guida pratica alla programmazione in BASIC per gli Atari ST, aggiornata ai più recenti comandi del BASIC ST. Questo libro è destinato ai possessori e utenti di un ST, sia che si stiano accostando per la prima volta alla programmazione, sia che vogliano approfondire le caratteristiche peculiari degli Atari ST.

In una serie di unità concise e facili da seguire, si lavora passo passo con l'autore per progettare, analizzare e modificare una collezione di programmi BASIC che illustrano tutto, dai rudimenti della programmazione alle tecniche più avanzate. Ogni routine è un'unità utile, oltre che un esempio di stile di programmazione. Le routine presentate nel libro permettono di:

* creare effetti sonori
* scrivere musica per tre voci
* tracciare grafici tridimensionali
* scorrere sul monitor tutti i 512 colori
* generare istantaneamente grafici a istogrammi e diagrammi circolari
* progettare usi creativi per il mouse

...e molte altre cose ancora! Il capitolo "Programmi e problemi", inoltre, sviluppa un approccio per continuare a scrivere programmi che risolvono particolari esigenze personali degli utenti. Il testo contiene anche

* una dettagliata descrizione dell'hardware del 520ST e del 1040ST
* un'utilissima guida ai messaggi d'errore Atari
* una guida di consultazione delle parole chiave
* esaurienti note sull'installazione.

L. 42.000 **ISBN 88 7081 325 8**



ATARI® ST™ programmazione in BASIC

Tim Knight